Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Path planning and map monitoring for self-driving vehicles based on HD maps

Autor: Alejandro Díaz Díaz

Tutores: Luis Miguel Bergasa y Carlos Gómez Huélamo

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Path planning and map monitoring for self-driving vehicles based on HD maps

Autor: Alejandro Díaz Díaz

Directores: Luis Miguel Bergasa y Carlos Gómez Huélamo

Tribunal:

Presidente: Daniel Pizarro Pérez

Vocal 1º: Ignacio Parra Alonso

Vocal 2º: Luis M. Bergasa Pascual

Calificación: .....

Fecha: .....

"You can't stop the waves, but you can learn to surf."
Jon Kabat-Zinn

Acknowledgements

Este trabajo refleja mi paso estos dos últimos años por el máster y el grupo de investigación Robesafe.

No han sido unos años fáciles, y nunca tuve ninguna intención de hacer este máster o entrar en un grupo de investigación, pero no puedo estar más orgulloso de las decisiones que he tomado y me han llevado hasta este punto.

Lo primero como siempre, tengo que estar eternamente agradecido a mis padres y mi hermano por haberme dado la oportunidad seguir estudiando, apoyándome hasta llegar aquí. Por supuesto hay que tener en cuenta a mi querida Bulma, no se cómo habría aguantado estos meses de confinamiento sin ella.

A Luis Miguel por haber confiado en mi, dándome la oportunidad de entrar al grupo y guiándome en todo momento, junto con el resto de profesores. También tengo que agradecer en especial a Manuel el haberme ayudado con todas las dudas de LaTeX. Sonará a tópico, pero creo que tengo los mejores compañeros de grupo que podría tener. Javi, Carlos, Rodri, Felipe, Óscar, ... a pesar de los días buenos y los días no tan buenos, habéis sido un pilar fundamental como amigos y como compañeros. También tengo que agradecer los compañeros de máster que he tenido, que alguien me explique cómo podríamos haber aprobado las 14 asignaturas que llegamos a tener el primer año si no hubiese sido trabajando todos juntos.

Finalmente, tengo que agradecer a toda la gente de la que me he rodeado. ha formado y forma parte de mi día a día.

He comenzado diciendo que jamás me habría imaginado llegar hasta este punto, y la cuestión es que ahora mismo tampoco tengo muy claro hacia donde quiero ir. Pero una vez leí "En caso de duda, rema", así que aquí estoy, remando y feliz de lo que estoy consiguiendo. Espero en un futuro poder leer esto y reirme recordando lo poco claro que lo tenía.

Resumen

Este trabajo ha sido realizado dentro del contexto del proyecto Techs4AgeCar en el grupo de investigación Robesafe, cuyo objetivo es el desarrollo de un vehículo de conducción autónoma. Forma parte de dos líneas distintas del proyecto, la de mapeado y la de planificación, ya que ambas están directamente relacionadas.

Se ha desarrollado un planificador de rutas global basado en mapas de alta defición (HD Maps) offline previamente generados.

Por otro lado, también se ha cubierto toda la parte de generación de mapas que posteriormente son utilizados por el planificador.

Además, se ha desarrollado un módulo capaz de aprovechar la información proporcionado por el mapa, de forma que se monitorizan los elementos relevantes y cercanos al coche que afectan a la ruta, como son carriles, intersecciones y elementos regulatorios.

Palabras clave: Planificación global; Mapa HD; Simulador Carla; OpenDRIVE (XODR); Robot Operating System (ROS).

Abstract

This work has been done within the context of the Techs4AgeCar project in the Robesafe research group, whose project focuses on the development of an autonomous driving vehicle. This work is part of two different layers of the project, mapping and planning layers, since both are directly related.

A global route planner has been developed based on previously generated offline HD Maps.

Therefore, the entire part of generating maps that are later used by the planner has also been covered.

In addition, a module capable of taking advantage of the information provided by the map has been developed, so that the relevant elements close to the vehicle that affect the route such as lanes, intersections and regulatory elements are monitored.

Keywords: Global Path Planning; HD Map; Carla Simulator; OpenDRIVE (XODR); Robot Operating System (ROS).

Contents

Resumen ix

Abstract xi

Contents xiii

List of Figures xvii

List of Tables xix

List of Acronyms xix

List of Symbols xix

1. Introduction 1

1.1. Motivation . . . 1

1.2. State of the art . . . 1

1.2.1. HD Maps . . . 1

1.2.1.1. Introduction to HD Maps . . . 1

1.2.1.2. What is an HD Map . . . 2

1.2.1.3. How to generate an offline HD Map . . . 2

1.2.1.4. Existing HD Maps solutions . . . 3

1.2.1.4.1. Open source solutions . . . 3

1.2.1.4.2. Commercial solutions . . . 5

1.2.1.4.3. Libraries to use HD Maps . . . 6

1.2.1.5. Pros and Cons of HD Maps . . . 7

1.2.2. Path Planning . . . 8

1.2.2.1. Introduction to Path Planning . . . 8

1.2.2.2. Path planning in robotics . . . 8

1.2.2.2.1. Global path planning . . . 8

Continuous maps . . . 8

Discrete maps . . . 8

1.2.2.2.2. Local path planning . . . 10
1.2.2.3. Path planning in AVs . . . 11
1.2.2.3.1. Global path planning . . . 12
Goal-directed . . . 12
Separator-based . . . 13
Hierarchical . . . 13
Bounded-hop . . . 14
1.2.2.3.2. Local path planning . . . 14
Graph search . . . 14
Sampling based . . . 14
Curve interpolation . . . 15
Numerical optimization . . . 15
Deep learning . . . 15

2. Techs4AgeCar **17**
2.1. Introduction . . . 17
2.2. Vehicle . . . 18
2.2.1. Hardware . . . 18
2.2.2. Software . . . 19
2.3. Architecture layers . . . 19
2.3.1. Mapping layer . . . 19
2.3.2. Planning layer . . . 20
2.3.3. Perception layer . . . 20
2.3.4. Decision making layer . . . 20
2.3.5. Control layer . . . 21
2.3.6. Localization layer . . . 21
2.3.7. Drive by wire layer . . . 21
2.3.8. Vehicle to user layer . . . 21

3. Theoretical Study **23**
3.1. Mapping . . . 23
3.1.1. Road map modelling . . . 23
3.1.1.1. Define roads using a reference line . . . 23
3.1.2. Lanes discretization . . . 23
3.1.3. Monitored area . . . 25
3.1.3.1. Braking distance model . . . 25
3.2. Planning . . . 26
3.2.1. Graph based path-finding algorithms . . . 26

3.2.2. BFS . . . 27
3.2.3. Dijkstra's . . . 28
3.2.4. A* . . . 28

4. Software and Technologies 31
4.1. Linux . . . 31
4.2. Python . . . 31
4.3. VSCode . . . 31
4.4. ROS . . . 31
4.5. RVIZ . . . 32
4.6. Docker . . . 32
4.7. Git . . . 33
4.8. OpenDRIVE . . . 33
4.8.1. General architecture . . . 33
4.8.1.1. Road . . . 34
4.8.1.2. Lane . . . 35
4.8.1.3. Junction . . . 37
4.9. Carla Simulator . . . 38
4.10. RoadRunner . . . 39

5. Development 41
5.1. Map Generation . . . 41
Data acquisition . . . 41
Map generation in Roadrunner . . . 41
Map import in Carla Simulator . . . 42
5.1.1. Why ASAM OpenDRIVE? . . . 42
5.2. Map Parser . . . 43
5.2.1. map_classes.py . . . 44
5.2.2. map_parser.py . . . 48
5.3. Map Monitor . . . 48
5.3.1. map_monitor.py . . . 49
5.3.1.1. Route Callback . . . 49
5.3.1.2. Location Callback . . . 50
5.3.2. monitor_visualizator.py . . . 51
5.3.3. map_visualizator.py . . . 52
5.4. Path Planner . . . 53
5.4.1. lane_graph_planner.py . . . 54
5.4.2. lane_waypoint_planner.py . . . 55
5.5. ROS Publishers and Subscribers . . . 56

6. Results **59**

6.0.1. Route in Carla Town03 . . . 60

6.0.2. Route in Campus UAH . . . 61

6.0.3. Overtaking with lane change . . . 63

7. Conclusions and future work **65**

7.1. Conclusions . . . 65

7.2. Future work . . . 66

Bibliography **67**

A. Budget **73**

A.1. Hardware . . . 73

A.2. Software . . . 73

A.3. Professional fees . . . 74

A.4. Execution budget . . . 74

A.5. Total costs . . . 74

List of Figures

1.1. Online dynamic HD map . . . 3
1.2. Offline static HD map . . . 3
1.3. Futuristic iCity with challenging 3D maps . . . 4
1.4. Map design software interfaces . . . 4
1.5. NVIDIA DRIVE Mapping . . . 5
1.6. HERE HD Live Map . . . 6
1.7. Lanelet right and left bounds . . . 6
1.8. Localization problem leads into a wrong route [1] . . . 7
1.9. Potential Field Map . . . 9
1.10. Cell maps . . . 10
1.11. Directed Graph Representation . . . 11
1.12. MDP and POMDP describing a typical RL configuration . . . 12
1.13. VFH: Polar obstacle density histogram . . . 13
1.14. Hierarchical vertex importance . . . 13
1.15. Graph based path planning algorithms . . . 14
1.16. RRT [2] . . . 15
1.17. Curve interpolation local planning techniques . . . 16

2.1. Platform evolution . . . 18
2.2. T4AC architecture layers . . . 19
2.3. Perception layer . . . 20
2.4. Localization layer . . . 21

3.1. Reference line composed by a sequence of geometries . . . 24
3.2. Monitored area in CARLA Simulator . . . 26
3.3. Connectivity graphs . . . 26
3.4. Graph based path planning algorithms . . . 27

4.1. RVIZ interface example . . . 32
4.2. Relation between OpenDRIVE, OpenCRG and OpenSCENARIO . . . 34

4.3. Parts of a road . . . 34
4.4. Elevation and lateral profile in a road . . . 35
4.5. Signals in OpenDRIVE . . . 36
4.6. A road with lane sections . . . 36
4.7. Lane links for road with id 10 . . . 37
4.8. Junction in OpenDRIVE . . . 37
4.9. CARLA Simulator . . . 38
4.10. RoadRunner interface . . . 39
4.11. RoadRunner complex junction example . . . 40
4.12. RoadRunner custom traffic sign example . . . 40
4.13. RoadRunner GIS image . . . 40

5.1. Map generation steps . . . 42
5.2. Overview of the main features for the different road geometry mapping models [8] . . . 43
5.3. Map Parser structure . . . 43
5.4. T4ac_Header class for Map Parser . . . 44
5.5. T4ac_Road class for Map Parser. Part 1 . . . 45
5.6. T4ac_Road class for Map Parser. Part 2 . . . 46
5.7. T4ac_Road class for Map Parser. Part 3 . . . 47
5.8. T4ac_Junction class for Map Parser . . . 48
5.9. Map Monitor structure . . . 49
5.10. Map monitor in Town03 . . . 52
5.11. Carla Simulator Town03 and its representation in RVIZ by Map Visualizator . . . 53
5.12. Height change in Town03 and its representation in RVIZ by Map Visualizator . . . 53
5.13. Path Planner structure . . . 54
5.14. Two roundabouts DiGraph using Netwokx . . . 55

6.1. Map generation and visualization from UAH campus image . . . 59
6.2. Top view of Town03 in CARLA Simulator (left) and RVIZ (right) . . . 60
6.3. Top view of route launched in Town03 . . . 60
6.4. Detail of areas in Figure 6.3 . . . 61
6.5. Top view of Campus UAH in CARLA Simulator (left) and RVIZ (right) . . . 62
6.6. Top view of route launched in Campus UAH . . . 62
6.7. Detail of areas in Figure 6.6 . . . 63
6.8. Overtaking with lane change . . . 64

List of Tables

5.1. Map visualizator ROS publishers 56
5.2. Map monitor ROS subscribers 56
5.3. Map monitor ROS publishers 56
5.4. Monitor visualizator ROS subscribers 56
5.5. Monitor visualizator ROS publishers 57
5.6. Path planner ROS subscribers 57
5.7. Path planner ROS publishers 57

A.1. Hardware 73
A.2. Software 73
A.3. Professional fees 74
A.4. Execution budget 74
A.5. Total costs 74

Chapter 1

Introduction

1.1. Motivation

This work has been done in the context of the T4AC project, that mainly consists in a project related to the development of a self-driving vehicle.

It is essential to have a path planning module in a self-driving vehicle. Before this proposal, there was another path planning module in the project. It was based on Lanelet and used OpenStreetMaps (OSM) manually generated with JOSM.

The group changed the simulator used in the project to Carla Simulator, because all the advantages it offers. Carla uses XODR map format, so a new planner had to be developed to work with XODR maps. This XODR file can be used in the real vehicle following a logic navigation architecture design scheduler, so these were the main reasons to develop a new planner:

- A planner that works in Carla (using XODR files)
- A planner that works the same way in both real and simulation cases (using XODR files in both cases)

Once we have moved all the mapping concept from OSM (OpenStreetMaps) to XODR (OpenDRIVE) format, a method for generating new maps must be set. The map generation have been solved using RoadRunner, that also represent a significant improvement with an ultra realistic representation of the map in the simulation.

A new concept has also been developed, taking advantage of the data that OpenDRIVE maps offer, that is the map monitor. It is a module that provides useful data of close elements (lanes, intersections and regulatory elements) to other modules, as for example, to the perception module, so it knows where to look for information.

1.2. State of the art

1.2.1. HD Maps

1.2.1.1. Introduction to HD Maps

An autonomous vehicle needs to locate itself in the environment to know what is happening close to it, in order to make decisions and execute a correct navigation similar to a human.

When we talk about location, the first thing we need is a map where to be located. Maps are something that have been part of our live since a long time ago, but if we focus on maps regarding vehicles, these are mainly road maps.

Anyone that has ever used a traditional road map knows how complex it can be. Nowadays this is not a problem, because most of the people use digital maps integrated on their smartphones or other electronic devices.

These kind of systems are based on localizing the device inside the vehicle using a Global Positioning System (GPS) and planning a route from the current position of the vehicle to a goal destination. In addition, GPS modern systems usually offer other extra information related to the route as real time information of the traffic or the possibility of adding an intermediate rest stop in the closer service area.

In most of the cases this information is more than enough to drive in a manual way, but not in the case of self driving vehicles.

Here is where the concept of HD maps appears.

1.2.1.2. What is an HD Map

An HD map is a specific type of map particularly designed for machines that contains a huge quantity of accurate metadata related to the road map. All this data is defined in a text file that can be processed by a computer. There is not an official definition about which data an HD map must include, but in the case of road maps, we could be talking about: roads, lanes, regulatory elements, topological information or any other information that can be considered relevant to drive in safety way.

This information can be used by different modules of a self driving vehicle, mainly the path planning module, but also can be used in other tasks, as for example, by the perception module to limit the area where to look for obstacles. HD mapping can be used to alleviate the computational load of other modules and add confidence to the environment model.

The combination of HD maps with onboard sensors and control systems leads to high-performance driving at the limits and is a very promising approach to enable autonomous driving [3].

The next level for HD maps is map generation, which is a very difficult task that can be tackled using two different approaches:

- 1) dynamic HD map on the cloud that is being constantly updated by other users in real time (Figure 1.1).
- 2) static offline HD map previously generated (Figure 1.2).

Current HD maps must be used as a tool by autonomous vehicles as traditional maps are used by humans, because current roads have being built for human users. But thinking on a future driving generation, intelligent driving infraestructures must be also considered. A concept of iCity, as shown in Figure 1.3, where Vehicle-to-vehicle (V2V) communication and Vehicle-to-infraestructure (V2I) communication exist [4], is a long-therm solution that could change the current idea of HD maps.

1.2.1.3. How to generate an offline HD Map

Currently does not exist a common standard about how an offline HD map must be generated, but there are multiple existing formats to do it. There are two main ways to generate it: manually or automatically.

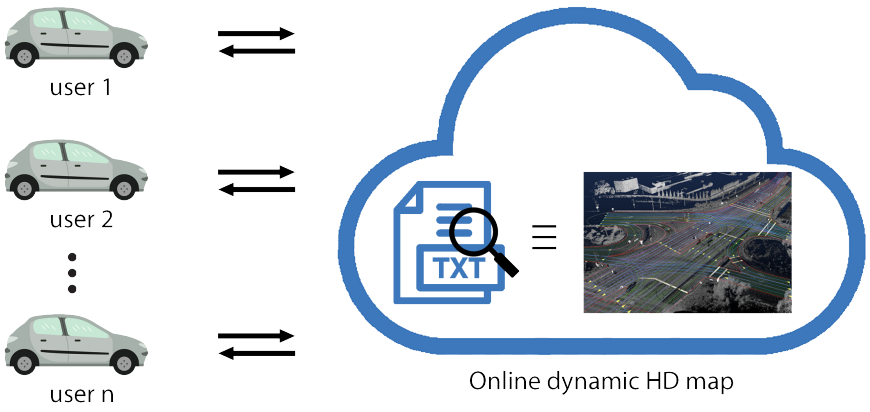


Figure 1.1: Online dynamic HD map

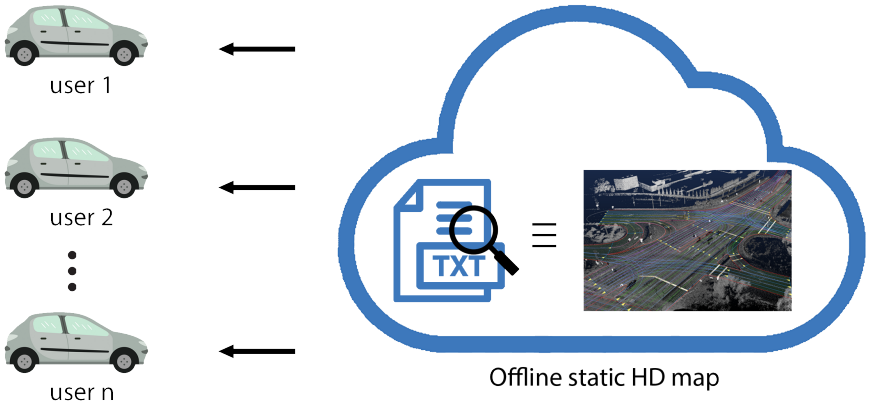


Figure 1.2: Offline static HD map

Automatically from converted scans of the real world roads is the optimal long-term and scalable solution [5] [6], but it can become a very complex task with high technical requirements depending on the map format in order to obtain a reliable method.

Manually is also a simpler option but requieres a lot of human work, which takes time and is susceptible to human errors. HD maps are defined in a text file, so it could just be created with a text editor if the format of the map is well known. But it may be too hard, so is better to use any of the existing HD map designing tools to facilitate the work, as JSOM or RoadRunner. In our case we have used RoadRunner [7], a software application with a graphical interface to generate HD maps in OpenDRIVE format 4.8. The reason is that HD maps in CARLA simulator, used in this work, have been developed by using this tool.

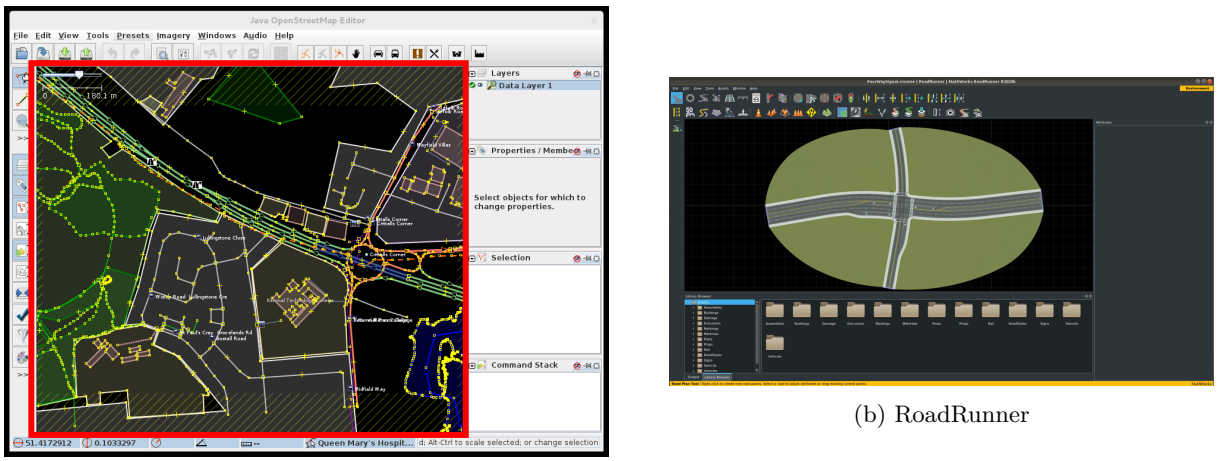
In Figure 1.4 are shown both interfaces for JOSM and RoadRunner.

1.2.1.4. Existing HD Maps solutions

As we have seen, there is not a common standard but multiple formats. We can classify them in open source formats and commercial solutions.

1.2.1.4.1. Open source solutions In [8] is presented a comparative of the most popular formats. We will focus on the two most relevant: **OSM (OpenStreetMaps [9])** and **XODR (ASAM OpenDRIVE**

Figure 1.3: Futuristic iCity with challenging 3D maps



(a) JOSM

(b) RoadRunner

Figure 1.4: Map design software interfaces

[10]).

OSM is a project initiated in 2004 by Steve Coast to create open source and editable maps. It is an editable map database built and maintained by volunteers and distributed under the Open Data Commons Open Database License. OSM maps can be downloaded from its web, or created using one of the multiple existing applications as JOSM (Java OpenStreetMap Editor [11]). This format allows to create maps using geographic points with different tags associated. Those tags are used to define their topology and how they are related to other elements.The map is modeled with 3 primitives: nodes, ways and relations. Nodes are the ones that represent those geographical coordinates, ways that consist on a list of nodes and relations that are formed by any number of members. Members may be any of the three elements and have a specific role. Other road properties are given as associated tags. OSM is good for representing topological information but has drawbacks when representing precise geometries of the roads because roads are only represented as lines (ways) that are formed by a list of nodes.

XODR is a standard for the description of road networks, modeling the road network along a reference line. Versions OpenDRIVE V1.4 and V1.5 were published by VIRES Simulationstechnologie GmbH, and

originally was an open source standard. In 2018 [12], the company ASAM was entrusted with the further development of OpenDRIVE, releasing their own version under the name of ASAM OpenDRIVE V1.6. In this document we will refer to this format in both ways, both OpenDRIVE or ASAM OpenDRIVE without specifying the version. Currently, the standard and all the documentation are still available free of charge on their web. OpenDRIVE V1.4 is used in Carla Simulator [13] and in our project. Maps generated with the application RoadRunner are in version V1.4.

1.2.1.4.2. Commercial solutions Mapping companies are also working on commercial solutions working on HD maps as is the case of NVIDIA DRIVE Mapping module [14], part of their End-to-End Autopilot Systems solution and the solution of HERE HD Live Map [15].

The module **NVIDIA DRIVE Mapping** is an open and scalable solution that combines a sensor suite, software and software APIs, with high-definition maps provided by mapping companies.

It consists of:

- DRIVE Localization for determining the precise 6-DOF position and orientation of an autonomous vehicle within an HD map with centimeter level accuracy.
- Drive Mapstream for updating cloud based HD maps with road features perceived by DRIVE Perception.
- Drive MyRoute for creating crowdsource maps of urban areas where there are not HD maps yet.

In Figure 1.5 we can see the two main parts: one in the autonomous vehicle (DRIVE Localization and MapStream Client), and the other in the cloud (DRIVE MapStream and DRIVE MapServices).

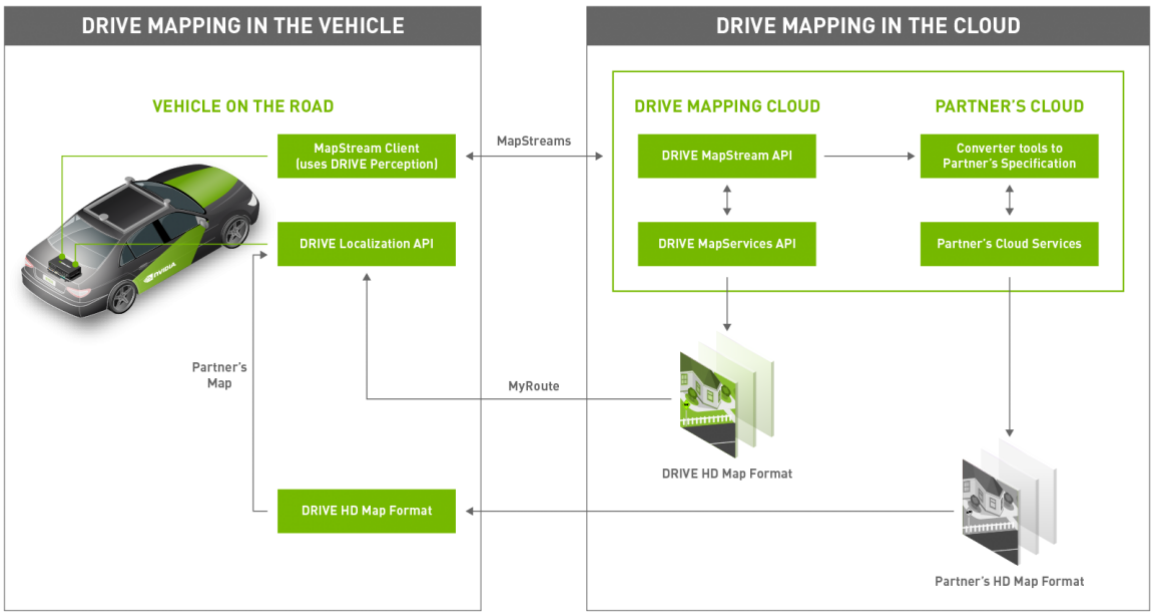


Figure 1.5: NVIDIA DRIVE Mapping

The other commercial solution mentioned is **HERE HD Live Map** by HERE Technologies. It uses machine learning to validate map data against the real world in real time (Figure 1.6). Their self-healing map analyzes data from multiple sources, such as satellite imagery and sensors from OEM fleets in real-time, allowing the maps to always stay fresh and reliable.

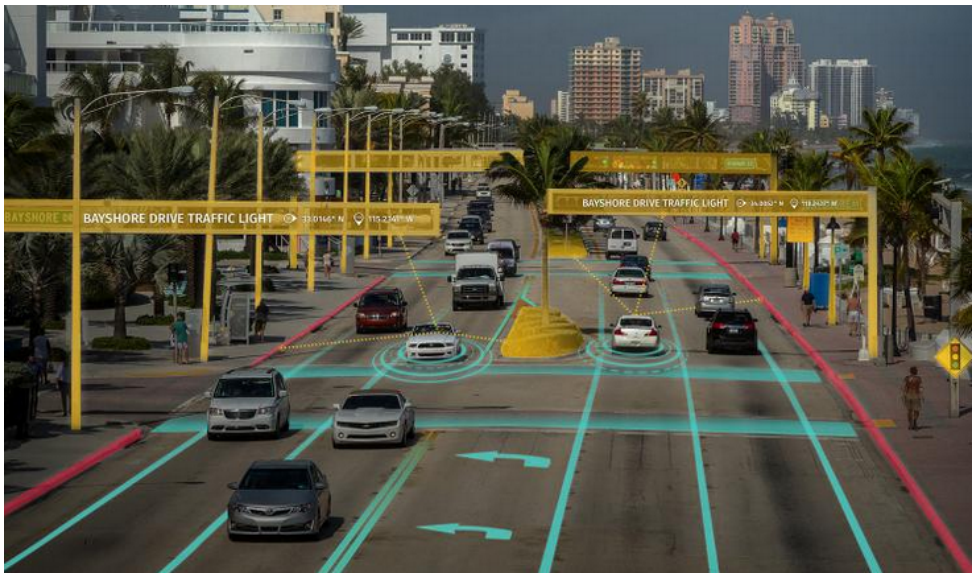


Figure 1.6: HERE HD Live Map

1.2.1.4.3. Libraries to use HD Maps Once we have an HD map file, some library is needed to translate that information in useful data for other modules of the system.

The problem of not having a common standard map format also affects the idea of creating a common standard library that may be used to process HD maps data. In most of the cases every project ends making its own little library for its own format.

Nevertheless, there are some existing libraries to be used with the existing open formats. We will see one of the most popular *libLanelet* [16] and the one used in our project *libCarla* (that can be used through its API [17]).

Lanelet [18] is a library that can be used for OSM maps. It is a concept for map representation presented in 2014. It models the relevant parts of the environments by means of lanelet elements, which are atomic, interconnected drivable road segments, geometrically represented by a left and right bounds (Figure 1.7). The bound is approximated by a list of points, yielding a polygonal line or a polyline. A set of elements is identified to describe traffic regulations and attribute those elements to the lanelets. Lanelets are concatenated in driving corridors. A driving corridor is a set of adjacent lanelets. A road graph can be built using lanelets to then using some graph routing algorithm.

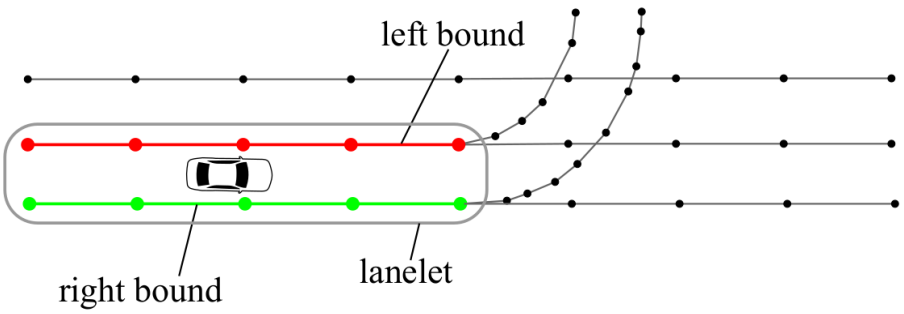


Figure 1.7: Lanelet right and left bounds

Along with the publication where Lanelets are presented, a C++ library libLanelet is also provided by the authors to read and parse XML files and put into action their method. Lanelet approach can be

implemented within OSM map format. In 2018, a new version of Lanelet was presented in [19]. In [20] are explained the differences between lanelet2 and lanelet1.

LibCarla is a library released by Carla Simulator developers within the simulator framework, that can be used for XODR maps. It has been originally developed in C++ and presented with a documented PythonAPI [17]. This library can be easily used through their API to manage XODR maps, in addition to other applications related to the simulator, but the possibilities that it offers are limited. That is why we have developed our own code modules within this library.

1.2.1.5. Pros and Cons of HD Maps

After having seen in detail everything related to HD maps, we are finally going to comment the pros and cons of suing HD maps.

Pros:

- Reduce computational load of other modules
- Add confidence to the system , as it does not have occlusion problem
- Provide global path planning

Cons:

- Localization error can lead into a wrong route (Figure 1.8)
- There is not a complete and robust standard
- Maps must be previously generated and constantly updated

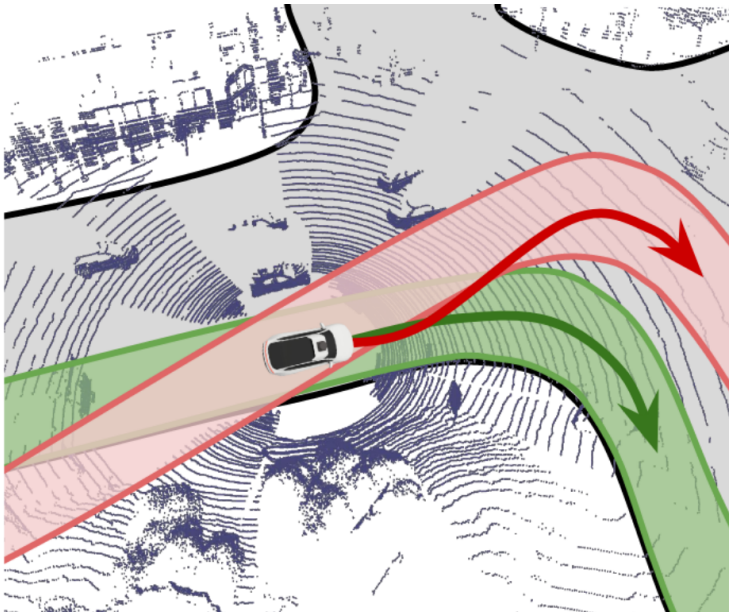


Figure 1.8: Localization problem leads into a wrong route [1]

1.2.2. Path Planning

1.2.2.1. Introduction to Path Planning

Path planning is a fundamental part of an AV, as it is in charge of getting the route that the vehicle must follow. This route is usually sent to the control module, or in some cases the control module is already integrated with the path planning module.

There are multiple approaches and multiple ways to classify path planning algorithms. For example if the environment is known or unknown, if the driving area is structured or not, but the main characteristic to classify them is local and global planning algorithms.

First, we are going to talk about path planning in robotics and then we will focus on path planning in Autonomous Vehicles, the interest of this survey.

1.2.2.2. Path planning in robotics

1.2.2.2.1. Global path planning Global planning is the decision process to get a goal destination. These decisions are usually based on a map of the environment. Navigation maps can be classified in multiple ways, but here we will break it down into two main groups: continuous and discrete maps.

Continuous maps These are geometric maps with a continuous definition from a starting to an end point, without any gap between them. In some cases discrete a map can be modeled in a way that it becomes a continuous map. Multiple path planning algorithms can be applied to continuous maps, some of them are:

- **Visibility Graph (V-graph):** This approach is about the aim of finding the shortest path of a point among polygonal obstacles. In [21] was proposed the classical method to deal with this problem. The algorithm transforms the obstacles so that they represent the locus of forbidden positions for an arbitrary reference point on the moving object. A trajectory of this reference point which avoids all forbidden regions is free of collisions. Trajectories are found by searching a network which indicates, for each vertex in the transformed obstacles, which other vertices can be reached safely. In [22] a Dynamic Visibility Graph (DVG) method was presented to improve the efficiency of V-graph with only considering local region.
- **Virtual Field Force (VFF):** The robot is represented as a point affected by the influence of a potential field. Goal generates attraction force and obstacles generate repulsive force, defining the forces as mathematical functions (Figure 1.9). The robot's movements are similar to a ball rolling downhill towards the goal. In [23] a path planning method is presented using a potential field representation of obstacles, combining in two levels global and local planning.
- **Voronoi Diagrams:** Some path planning methods are based on these kind of diagrams that are a partition of a plane into regions close to each of a given set of objects. In [24] a technique based on Voronoi Diagrams is developed to calculate an optimal path from a start to a goal location in presence of simple disjoint polygonal obstacles in the plane, evaluating Euclidean path length, smoothness and clearance from obstacles.

Discrete maps Discrete maps are defined using a limited set of elements, which means less computational load regarding to continuous maps. Discrete maps can be modeled using cells or in a topological way.

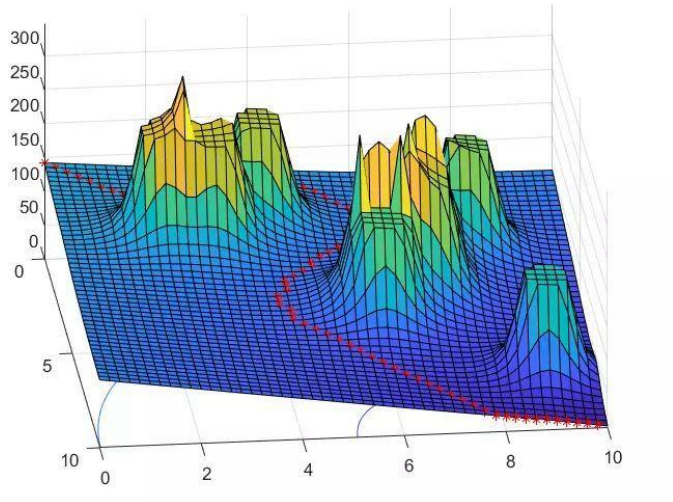


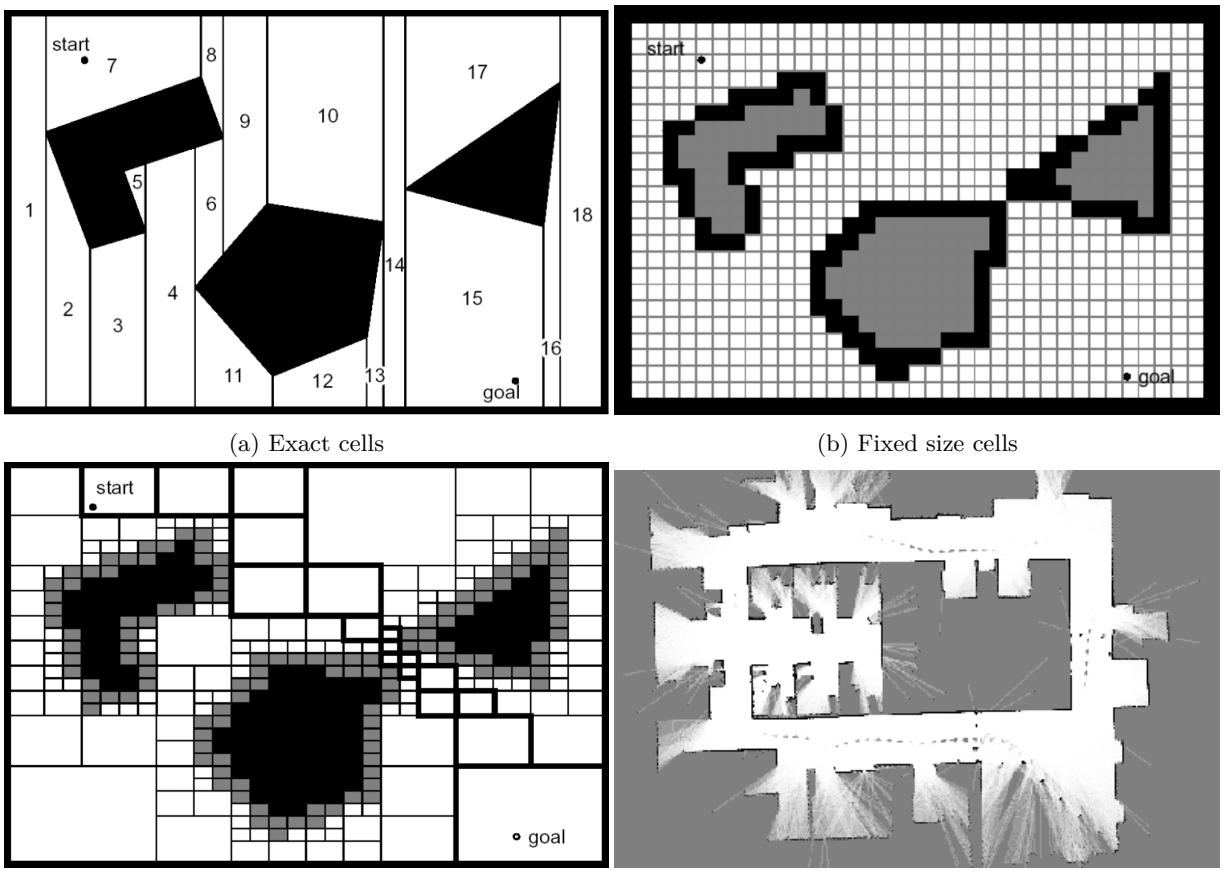
Figure 1.9: Potential Field Map

Cell maps are based on decomposing the map into cells. These cells are connected to create a connectivity graph, then path planning algorithms can be applied to this graph. Depending on the cell shape and size, there are different types of cell maps:

- **Exact cells** (Figure 1.10a) : Non occupied space is covered using polygons. It is not important the position of the robot in the free area but the ability to move from one to another. The relations between the different areas are stored using connectivity graphs. This connectivity graphs can be used to apply path planning algorithms such as: Bread First Search (BFS), A*, Dijkstra.
- **Fixed size cells** (Figure 1.10b) : The map is decomposed in fixed size cells which values can be 0, 1 or unknown. The problem with this approach is that depending of the chosen size, too narrow path might not be considered. Algorithms that can be applied to this type of maps are: Wavefront using a Gradient Path Planning, BFS, A*, Dijkstra.
- **Adaptive size cells** (Figure 1.10c) : It solves the problem of narrow paths starting from a fixed size for non occupied areas that becomes reduced until a maximum resolution. Algorithms that can be applied to this type of maps are: BFS, A*, Dijkstra.
- **Occupancy grid** (Figure 1.10d) : Cells have a minimum fixed size and a weighted probability of being occupied that grows with each impact received by the sensors. These sensors are usually laser sensors. Algorithms that can be applied to this type of maps are: Wavefront, BFS, A*, Dijkstra.

Topological maps avoid geometrical measurements of the environment. Instead, these maps focus on relevant characteristics of the environment. Two main approaches can be used for path planning in topological maps: deterministic methods using graphs theory or stochastic methods using decision theory.

- **Graph methods:** Graphs have two main elements: nodes and edges (Figure 1.11). Nodes are point interest areas and edges connect different nodes. If an edge connects two nodes, it means that a node can be reached from the other without going through any other node. In this type of topological representation nodes must no be separated a fixed distance, but if edges are weighted with distances the graph could become a discrete map. Then, path planning algorithms for connectivity graphs can be applied: BFS, A*, Dijkstra.



(a) Exact cells

(b) Fixed size cells

(c) Adaptive size cells

(d) Occupancy grid

Figure 1.10: Cell maps

- **Decision methods:** These methods, Markov Decision Processes (MDP) or Partially Observable Markov Decision Processes (POMDP) (Figure 1.12), are based on probabilistic estimates with a set of defined policies for decision making. It is about finding the optimal policy that maximizes the achievement of the objective.

1.2.2.2.2. Local path planning Local planning algorithms are also considered obstacle avoidance algorithms, because its objective is to avoid collisions between the robot and the environment. It is usually based on local maps generated and currently updated by the onboard sensors. Local planning tends to be an independent task. However, it must be efficient to:

- Global goal
- Current velocity and cinematic of the robot
- Onboard sensors
- Current and future collision risk

Local path planning is an ongoing developing field of research but these are some of the traditional algorithms that have been the base for modern approaches:

- **Vector Field Histogram (VFH)**[25]: This method represents the local environment of the robot in a 2DOF grid with a probability value for each cell. The representation is reduced to a 1DOF

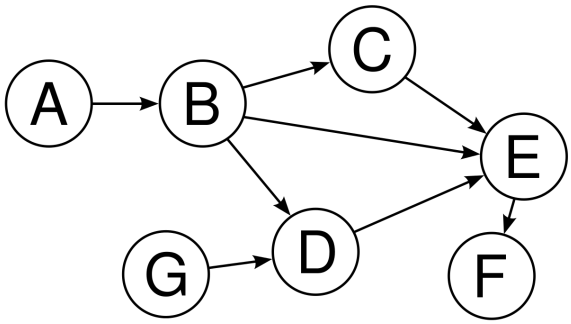


Figure 1.11: Directed Graph Representation

polar histogram (Figure 1.13). The histogram is calculated for every turning direction and every possible trajectory, then a cost function is used to select the best option. The original method was presented in 1991, and then further developments appeared. VFH+ [26] offers several improvements that result in smoother robot trajectories and greater reliability. VFH* [27] deals with situations that are problematic for purely local obstacle avoidance algorithms, verifying that a particular candidate direction guides the robot around an obstacle.

- **Curvature-Velocity Method (CVM)**[28]: This approach for local indoor mobile robots obstacle avoidance formulates the problem as one of constrained optimization in velocity space. Constraints that stem from physical limitations (velocities and accelerations) and the environment (the configuration of obstacles) are placed on the translational and rotational velocities of the robot. The robot chooses velocity commands that satisfy all the constraints and maximize an objective function that trades off speed, safety and goal direction.

- **Dynamic Window Approach (DWA)**[29]: This approach for collision avoidance is focused on robots equipped with syncho-drives. This approach incorporates the dynamics of the robot to reduce the search of space to the dynamic window, which consists of the velocities reachable within a short time interval. The dynamic window only considers admissible velocities yielding a trajectory on which the robot is able to stop safely. Among these velocities, the combination of traslational and rotational velocity is chosen by maximizing an objective function. In 1999 a new approach [30] solved the difficulties when reaching the goal destination adding a global planner to the original method.

- **Nearness Diagram Navigation (ND)**[31] This is a reactive collision avoidance method for vehicles that move in dense and complex scenarios. The situated-activity methodology is applied to design at symbolic level the reactive navigation method, based on the strategy "divide and conquer" to simplify the navigation difficulties. Many techniques could be used to implement this design for reactive method that are able to navigate in complex environments. In the paper is also presented a geometry-based implementation of their design. In [32] a new implementation is presented to achieve smoother trajectories.

1.2.2.3. Path planning in AVs

Path planning in Autonomous Vehicles is similar to path planning in general robotics but with some differences. Must be considered that automated driving is usually performed in controlled environments and there are other factors such as safety, comfort and energy optimization in addition to reaching the goal.

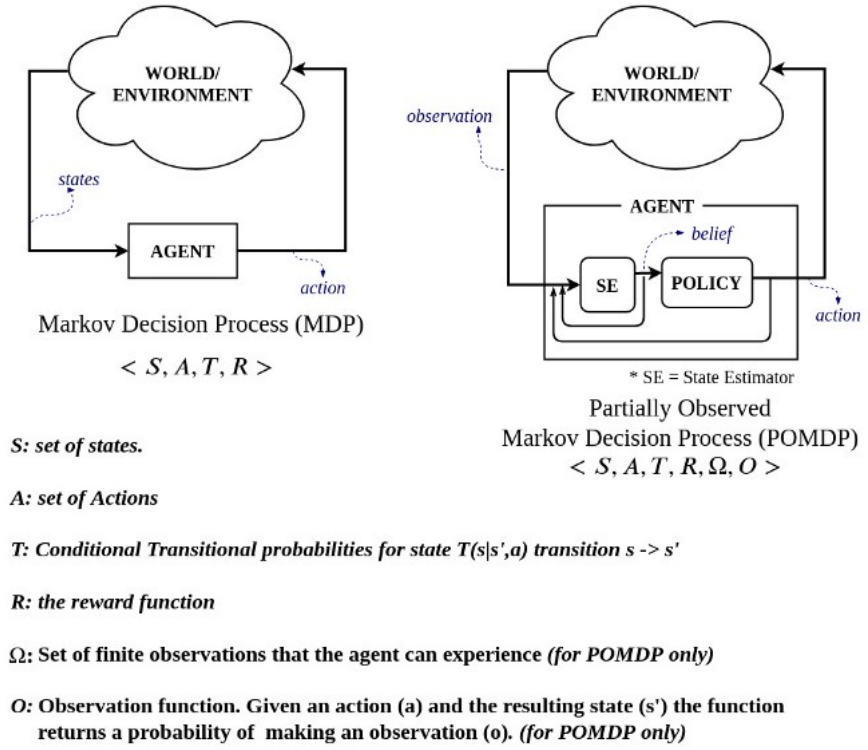


Figure 1.12: MDP and POMDP describing a typical RL configuration

Planning in AVs can be divided in two tasks too: global planning and local planning. In 1.2.2.2 we have been talking about local and global planning in robotics, now we will see the ongoing approaches to solve planning problem in autonomous vehicles.

1.2.2.3.1. Global path planning Global planning has the aim of finding a route from an initial to a destination point in the road network. Global path planning is something that has been well studied for years, since all of today's cars have a Global Positioning System (GPS) in order to localize its position in a road map and then calculate a route to a goal point. Advances in global planning algorithms can compute driving directions in milliseconds or less even at continental scale. There is a variety of techniques but they provide trade-offs between preprocessing effort, space requirements and query time. Some of the algorithms are focused on answering queries faster while others are able to deal with real-time situations considering traffic information. A first thorough survey was done in [33], but we will take as a base line the later extension of that survey in [34] that considers research published until January 2015. Recent non-traditional methods will be also mentioned after this survey of traditional methods.

Route planning traditional methods are examined under four categories:

- Goal-directed
- Separator-based
- Hierarchical
- Bounded-hop

Goal-directed Goal-directed method aim to guide the route search toward the target by avoiding the scans of vertices that are not in the direction of the goal, unlike Dijkstra's algorithm that scans all

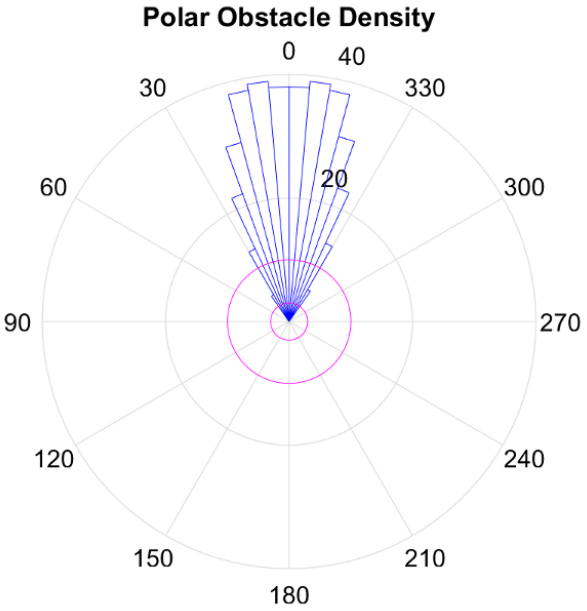


Figure 1.13: VFH: Polar obstacle density histogram

vertices with distances smaller that distances from origin to goal. A* [35] [36] algorithm is a standard goal-directed path planning algorithm and used extensively in various fields for almost 50 years [37].

Separator-based Although road networks are not planar graphs, they have small separators than can be used to decompose the graph in a simplifies graph. The aim of this technique is to remove a subset of vertices (Vertex Separators) [38] or arcs (Arc Separators) from the graph and compute an overlay over it. Using the simplified graph results in faster queries.

Hierarchical Hierarchical exploits the inherent hierarchy of the road network. Sufficiently long paths can be converged to a simplified small arterial network of main roads, such as highways. The algorithm only scans vertices of this subnetwork, speeding up the computational process. Despite the other algorithms find exact shortest paths, hierarchical type does not guarantee to find the exact path. Fu et al. [39] give an overview of early approaches using this method. [40] exploits the inherent hierarchical

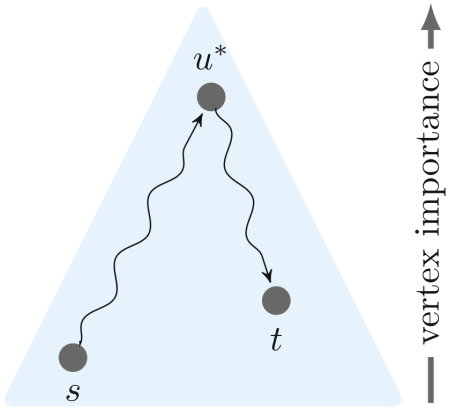


Figure 1.14: Hierarchical vertex importance

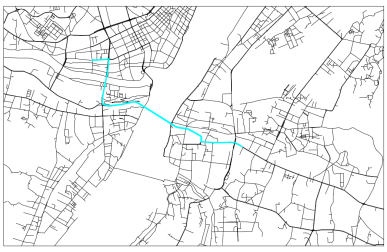
structure of road networks by adding shortcut edges.

Bounded-hop The basis of this idea is to precompute distances between pairs of vertices, implicitly adding virtual shortcuts to the graph connections. Bounded-Hop techniques use the precomputed distances between nodes instead of the input graph, returning the length of a virtual path with very few hops. One approach of Bounded-Hop techniques is Hub Labeling (HL) [41]. This is the fastest query time algorithm for path planning [34] in the expense of high storage usage.

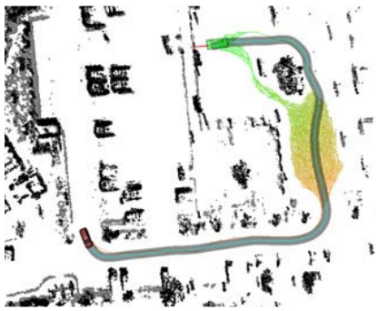
1.2.2.3.2. Local path planning Local planning aims to reach a point without collision and satisfying other criteria. These points may be provided by the global planner. According to [42], autonomous driving local planning techniques can be classified in 5 groups:

- Graph search
- Sampling based
- Curve interpolation
- Numerical optimization
- Deep learning

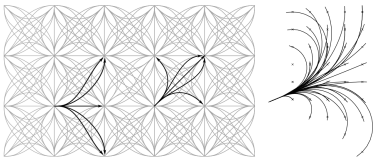
Graph search Graph search local planner are similar to graph search global planners. They use the same techniques such as Dijkstra's and A*. **A*** is a faster technique that **Dijkstra's** because it prioritizes one direction while Dijkstra's search in every direction. There are recent faster approaches, but some of them using as basis A*, such as the dynamic A* (D*) [43], Field D* [44], Theta* [45], Anytime Repairing A* and Anytime D* [46], among others. A more advanced graph planner is the **state lattice** algorithm. It uses a discrete representation of the planning area with a grid of states. This grid is referred as state lattice over of which the motion planning search is applied [47].



(a) Dijkstra's [48]



(b) A* [49]



(c) Lattices [50]

Figure 1.15: Graph based path planning algorithms

Sampling based Sampling-based methods offer an efficient solution solving timing constrains that deterministic methods can not meet. A detailed survey of sampling-based planners (SBP) can be found in [51]. This method is based on randomly sampling the configuration space, looking for connectivity inside it. The most popular approaches of SBP are the Probabilistic Roadmap Method (PRM) [52] and the Rapidly-exploring Random Tree (RRT Figure 1.16) [53].

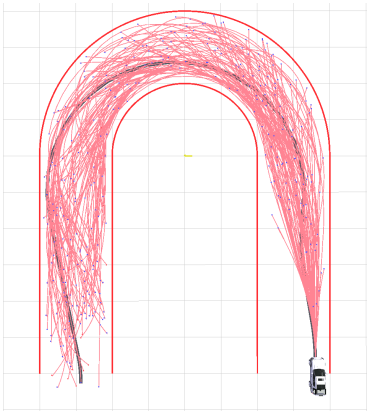


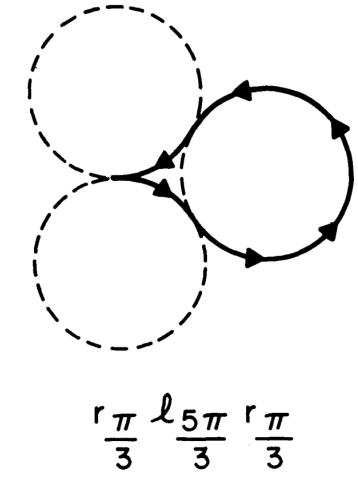
Figure 1.16: RRT [2]

Curve interpolation These algorithms take a previously set of way-points and generate a new set of data in benefit of the route aim: a smoother path, trajectory continuity and in the presence of obstacles, it suffices to compute a new path to avoid the obstacle and then re-entry the planned path. The most common curve interpolation algorithms are explained below and shown in Figure 1.17:

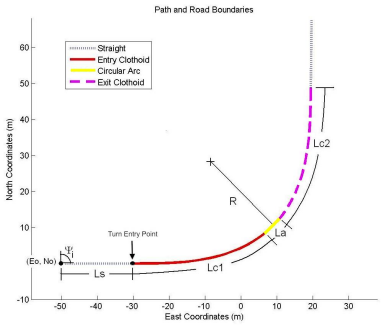
- **Line and circle**: Road fitting and interpolation of known waypoints. The planning problem in car-like robots has been approached using this method in [54] and [55]
- **Clothoid curves**: These curves are defined in terms of Fresnel integrals [56]. Clothoid curves curvature is equal to their arc-length which allows to define trajectories with linear changes in curvature, making smooth transitions between curved segments and straight ones and vice versa. This technique has been implemented in both general robotics and car-like robots [57].
- **Polynomial curves**: These curves are usually implemented to solve the constrains (angle and curvature) needed and fitting position in the interpolated points. These curves have been useful for lane change scenarios [58] and generating save trajectories for overtaking maneuvers [59].
- **Bèzier curves**: These curves select the optimal control points location to define their shape. Bèzier curves are based on Bernstein polynomials and the advantages of these curves are their low computational cost and the intuitive manipulation of the curve thanks to the control points that define it. Continuous concatenations of curves are possible [60] (Suitable for comfort).
- **Spline curves**: A spline is a piecewise polynomial parametric curve divided in sub-intervals that can be defined as polynomial curves [61], [62], b-splines [63], [64] (that can also be represented in Bèzier curves [65]) or clothoid curves [57]. These curves have low computational cost and the result is a general and continuous curvature path controlled by different knots.

Numerical optimization Trajectory generation optimizing parameters such as speed, steering speed, rollover constrains, lateral accelerations, jerk (lateral comfort optimization), among others. This method is often used to compute trajectories from kinematic contraints [69] and also to smooth previously computed trajectories as in [70]. Function optimization is a technique to find real valued function's roots.

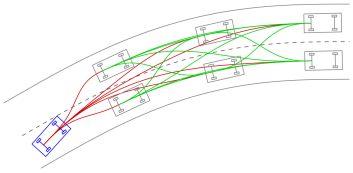
Deep learning Deep Learning (DL) and Reinforcement Learning (RL) are emerging alternatives to traditional path planning approaches. Fully convolutional 3D neural networks are able to generate future route paths using sensors input such as camera image or lidar point clouds [71]. In the case of RL,



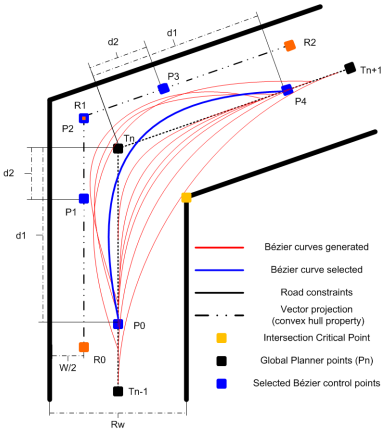
(a) Line and Circle [54]



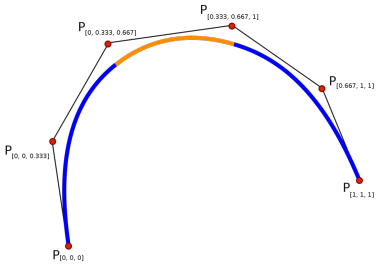
(b) Clothoid [66]



(c) Polynomial [67]



(d) Bèzier [68]



(e) Spline

Figure 1.17: Curve interpolation local planning techniques

the learning is done using input examples instead of raw data. These DL based planning techniques are promising, but they are not widely used in real-world systems yet.

Chapter 2

Techs4AgeCar

2.1. Introduction

Statistics show that 68% of the population in the European Union (EU), including associated states, is living in urban areas. According to the World Health Organization, nearly one third of the world population will live in cities by 2030. Aware of this problem, the Transport White Paper published by the European Commission in 2011 indicated that new forms of mobility should be proposed to provide sustainable solutions for people and goods safely. Regarding safety, it sets the ambitious goal of halving the overall number of road deaths in the EU between 2010-2020. However, the goal will not be easy, only in 2014 more than 25,700 people died on the roads in the EU (18% reduction). Besides, some studies show that fatal accidents increase with age for people 65 years and older.

Autonomous driving is considered as one of the solutions to the before mentioned problems and one of the great challenges of the automotive industry today. The existence of reliable and economically affordable autonomous vehicles will create a huge impact on society affecting environmental, demographic, social and economic aspects. In particular, it is estimated to cause a reduction in road deaths, improved traffic flow, reduced fuel consumption and harmful emissions associated, as well as an improvement in the overall driver comfort and mobility in groups with impaired faculties, like the elderly or disabled people.

Autonomous driving has attracted much attention recently by the research groups and industry, due to the billboards of various companies on expectations of market entry. However, his predictions seem to be very optimistic. A more scientific organization, such as IEEE, has recently predicted that by 2040 the majority of vehicles traveling on highways will be autonomous. Driving in urban environments will take longer, due to its complexity and uncertainty.

With this background, this project will focus on the research in technologies for the development of an autonomous electric car to assist the elderly population in predominantly urban environments. The proposal is disruptive because it raises new techniques applied on a future electric car, where Spain is an international reference, and targeted to a sector of the population with growing needs. The proposed system is based on the most advanced techniques of sensory perception and in-vehicle mapping, control and path planning in dynamic and changing environments.

A sensor fusion architecture based primarily on computer vision, laser and GPS technologies will be developed. The architecture will allow the mapping of the environment, the semantic classification of the scene and the real-time detection of obstacles in the path. This, together with vehicle positioning technology based on digital maps, will establish the planning and control algorithms to carry out the independent movement of the vehicle from a source to a destination provided by the user. Additionally,

a study of the behavior of senior drivers will take place, to identify shortcomings and limitations, and develop a customized HMI (Human Machine Interface).

2.2. Vehicle

2.2.1. Hardware

There are different commercial solutions for getting an automatized vehicle, but due to that they are very costly and work as a black box not having full control on it, it was decided to automatize an open source chassis solution.

The platform used as the mechanical base for this project is TABBY EVO, developed by the American company Open Motors, and was acquired in 2017 by the research group.

TABBY EVO is an open source platform and all the mechanical and electrical blueprints are available to be downloaded and modified. The initial equipment of the platform was: 4 seats, an asynchronous AC motor of 19kW, an AC-L1 controller from the Italian group SME, a BMS SCC24 from the XBW brand and a charger ZIVAN NG3 delivering 80V and 25A to batteries.

In a first upgrade of the platform, were added a battery pack and a tubular structure to mount the sensors. In a second upgrade, a steel body was designed for the platform, the mechanical steering column was replaced by an electrical one and all the hardware required for autonomous navigation was added. All the sensors are located in a roof rack, both location and perception sensors.

In Figure 2.1 is shown the evolution of the platform.

Figure 2.1: Platform evolution

The different processes to carry out the autonomous navigation are distributed in multiple systems along the vehicle: 3 Raspberry Pi, 1 NVIDIA Jetson AGX Xavier and 1 laptop MSI GT62VR 6RE (Dominator Pro 4K).

The sensors located in the system are:

- 1 LIDAR: Velodyne LIDAR VLP-16

- 1 Stereo Camera: ZED
- 1 GNSS
- 2 Optical incremental encoder

2.2.2. Software

The project has been developed in Linux and is structured in a branched architecture with 8 layers, some of them containing more sub-layers. Layers are defined as Git repositories to keep an easier version control and ROS is used to communicate the different processes.

The whole project is contained inside a docker file to guarantee an stable and self-dependent environment.

2.3. Architecture layers

Our autonomous navigation architecture is based on ROS, using launch files to run multiple ROS nodes that execute different and asynchronous processes that are communicated by ROS topics.

Each layer is focused on a task and can use different inputs from the sensors or from other layers. The 8 different layers are: Mapping, Planning, Perception, Decision Making, Control, Localization, Drive by Wire and Vehicle to User (Figure 2.2).

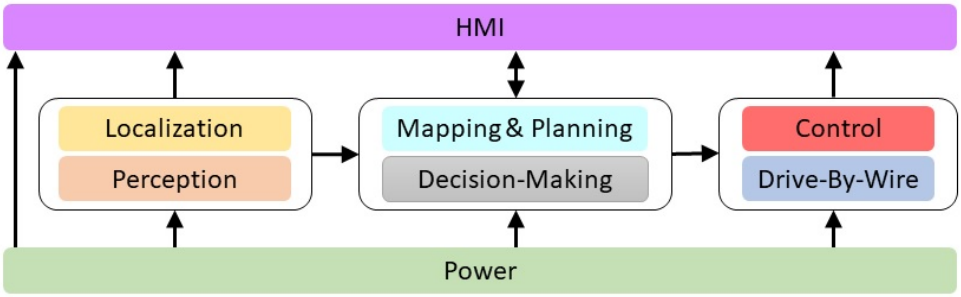


Figure 2.2: T4AC architecture layers

2.3.1. Mapping layer

One of our framework strengths is its portability between simulation and real environment. A map based on the University Campus has been designed using the RoadRunner application. An unique text file describing the road map can be used by the mapping layer both in a real case or in the simulator to obtain relevant information of the map to be processed.

ASAM Open DRIVE is the standard used to define the map as it provides geometrical and topological information about the roads and its relevant elements.

Inputs to this layer are the OpenDRIVE map as an .XODR file, current position of the vehicle and the route as a list of waypoints. Outputs are the monitorized lanes, intersections and regulatory elements close to the vehicle and affecting the route. Map lanes and the monitorized area are represented on the ROS visualizer RVIZ using marker messages. It also returns the map file parsed so it can be used by other layers.

2.3.2. Planning layer

Planning layer receives the map parsed by the mapping layer and creates a directed graph describing the road map. Once the graph road map is generated, every time a goal position is received the path planning is solved applying the graph-based method A* from the current position of the vehicle. A new route can be calculated at any time to change the goal or to perform a lane change.

2.3.3. Perception layer

In terms of environment perception, we carry out tracking-by-detection [72] using as input Bird's Eye View objects obtained through the sensor fusion of depth map and stereo RGB camera information. Despite the fact that in our real-world prototype we perform a late sensor fusion between the BEV stereo camera objects and the BEV laser objects branch, at the moment of writing this article CARLA does not offer enough quality in its laser pointcloud, even increasing the number of channels and points per second, so as to obtain similar results that in real-world datasets. Then, this set of BEV obstacles feeds a simple yet accurate tracking pipeline made up by a BEV Kalman Filter [73], that predicts the state of the objects from the current frame and update the object state based on the detected bounding boxes at current frame, and a Khun-Munkres (a.k.a Hungarian algorithm [74]) that associates the predicted trajectories with current detections. Finally, a B/D (Birth and Death) memory deals with the newly appeared trajectories (those matched trajectories that exceeds f_{min} frames) and dissapeared trajectories (unmatched trajectories exceeding age max frames).

Figure 2.3 illustrates the architecture of perception layer.

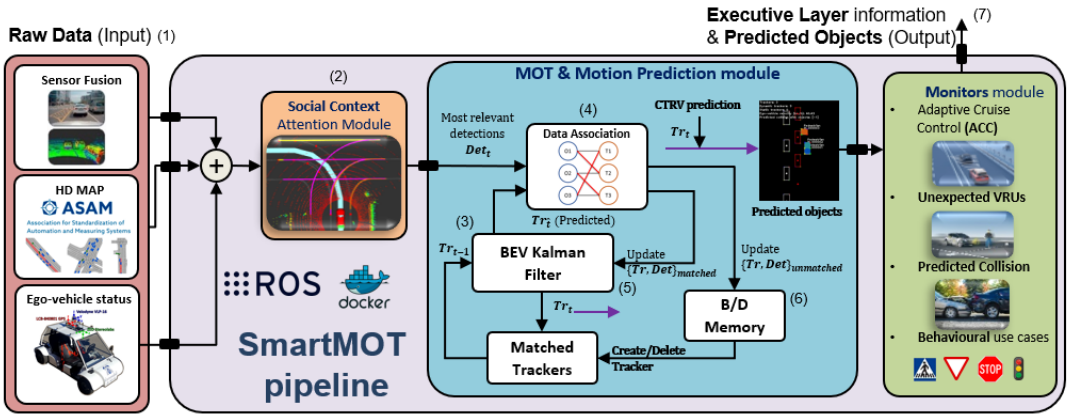


Figure 2.3: Perception layer

2.3.4. Decision making layer

The decision making module evaluates the information provided by the perception module, the actual state of the vehicle and the HD map features to select the adequate behaviour for each driving situation. We define two types of behaviours: Background behaviours, just as the unexpected pedestrian or the adaptive cruise control (ACC). These are continuously running, expecting a close vehicle or pedestrian to reduce the vehicle speed in a safe way. Standard use cases, as might be a stop, a give way, an overtaking or a crosswalk. This module is done using Petri Nets. For more information, the reader is referred to the publication [75].

2.3.5. Control layer

Our controller [76] performs a smooth interpolation of the waypoints given by the planner. Before the navigation starts, a velocity profile is generated using the curvature radius of each trajectory section. During the navigation the speed command is adjusted using this profile and the steer command is set using LQR techniques to ensure the trajectory tracking. The existing delays in the localization module and the vehicle actuators is compensated in the control loop.

2.3.6. Localization layer

This module is in charge of positioning the vehicle on a map with centimeter precision and in real time. Accurate and robust localization is a primary task for autonomous vehicle navigation. In a real environment, we estimate the vehicle's pose using the fusion of data provided by a differential GNSS (Global Navigation Satellite System) and by an encoder- based odometry. To improve the accuracy and reliability of the GNSS system, differential positioning techniques (DGNSS) and Real Time Kinematics (RTK) were used [77]. On the other hand, in simulation, CARLA provides the exact location of the vehicle via ROS, so a localization module is not required.

Figure 2.4 illustrates the architecture of perception layer.

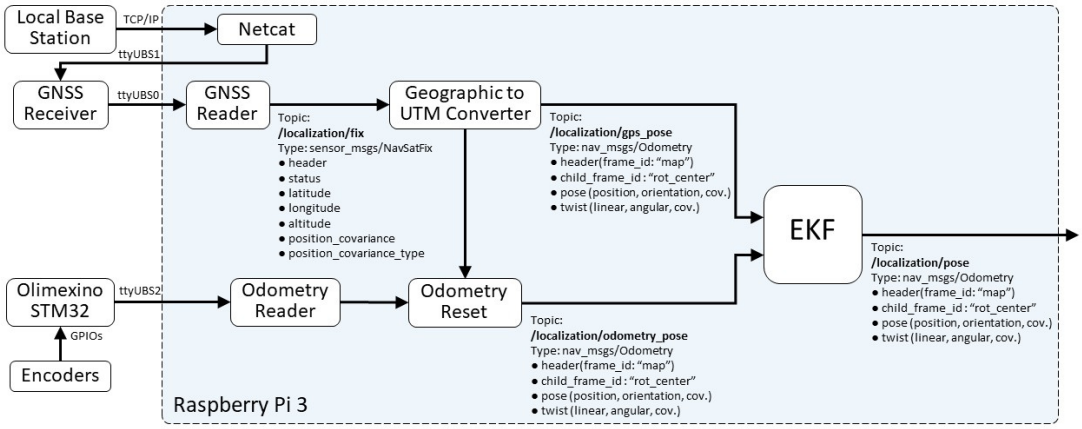


Figure 2.4: Localization layer

2.3.7. Drive by wire layer

The Drive-By-Wire module, which is implemented in the real vehicle, receives the steer and speed commands sent by the Control module and generates the electric signals to feed the electric motor and the steering wheel [78]. We have implemented PI controller that equals this module in simulation. This module receives the controller output and generates normalized throttle, brake and steer commands, which are the input signals required by the simulated vehicle.

2.3.8. Vehicle to user layer

Vehicle to User (V2U) layer is in charge of the interaction between the driver and the vehicle. This layer gives the opportunity of a new abstract level to the autonomous driving problem. Inside this layer, there are different systems that are used in the project. Firstly, there is the gaze focalization system that allows to monitor the gaze of the driver to know where he is looking at. With this system, we are able

to develop other tools that allows the transition manual/autonomous in a safety way or even develop an attention dataset that brings the opportunity to build an attention module that predict where the driver will focus his attention for a better scene understanding and in this way to perform a more natural driving behavior. Also, there are other systems in the vehicle that are part of this interaction, which allows the communication between the user and the vehicle.

Chapter 3

Theoretical Study

3.1. Mapping

3.1.1. Road map modelling

Once we have talked about HD maps, we are going to see how those maps can be modelled ir order to be defined in a text file.

There are different approaches for map modelling, but the main ones are:

- **Reference line:** Define roads along a reference line. This is the one used in XODR and we will see it in more detail.
- **Coordinate nodes:** Define road maps using points with properties associated. This is the method used in OSM, and its main disadvantage is that it must be complemented with some HD map framework as Lanelet.

3.1.1.1. Define roads using a reference line

This is the approach used in ASAM OpenDRIVE standard. How this format is structured will be treated deeper in next chapter, but in this section we will focus on how roads are modelled. In this case, the road is generated from a sequence of segments defined by its reference line. The reference line is a virtual line that can be composed by a sequence of different geometries: straight, arc or spiral as it is depicted in Figure 3.1. All geometries that describe the road shape and other properties of the road are defined along the reference line. These properties are lanes, signals and other road elements.

When using the software application RoadRunner, only straight and curve geometries are considered.

The reference line runs in s-direction while the lateral deviation from the reference line runs in t-direction. Must be considered that the s-direction does not indicate the driving direction, actually a reference line can define a bidirectional road.

3.1.2. Lanes discretization

The reference line is discretized as a list of points along the total length of the line separated by a constant value:

$$reference\,line = [0, d, 2d, ..., nd] \quad \forall \quad n = [\frac{total\_lenght}{d}] \tag{3.1}$$

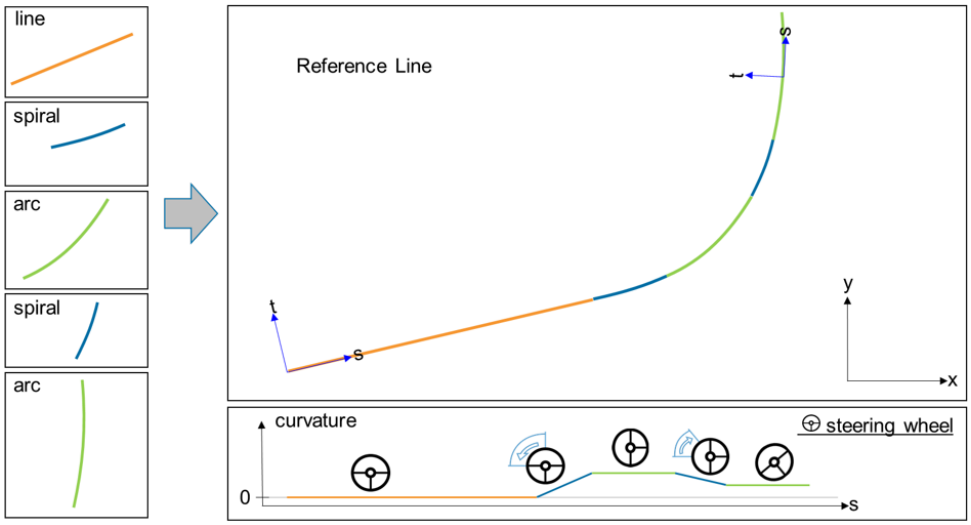


Figure 3.1: Reference line composed by a sequence of geometries

Where d is the distance between points and n the number of points.

The parameters needed to calculate it are:

- **x0**: Coordinate x where the reference line starts
- **y0**: Coordinate y where the reference line starts
- **hdg**: Heading at the beginning of the reference line
- **s**: Distance along the reference line
- **type**: Straight line or arc
- **central_node**: xyz of the node in the reference line to discretize lane boundaries
- **lane_width**: Width of the lane of the central node
- **curvature**: Only needed in case of arcs

To include height in the model, other metadata from the XODR file can be used but it will not be considered in this approach.

In case of **straight lines** these are the equations that define the road:

$$x = x_0 + (s \cdot cos(hdg)) \tag{3.2}$$

$$y = y_0 + (s \cdot sin(hdg)) \tag{3.3}$$

In case of **arc lines** the equation will be:

$$radius = \frac{1}{curvature} \tag{3.4}$$

$$hdg_{aux} = hdg + s \cdot curvature \tag{3.5}$$

$$x = x_0 + radius \cdot (cos(hdg + \frac{\pi}{2}) - cos(hdg_{aux} + \frac{\pi}{2})) \tag{3.6}$$

$$y = y_0 + radius \cdot (sin(hdg + \frac{\pi}{2}) - sin(hdg_{aux} + \frac{\pi}{2})) \tag{3.7}$$

Lanes are modelled generating points at both sides of the the discretized reference line, left_node and right_node.

Lane boundaries can be obtained using these equations for each central node:

$$\alpha_{degrees} = 90,0 - central\_node.yaw \tag{3.8}$$

$$right\_node.x = central\_node.x + cos(\alpha_{radians}) \cdot \frac{lane\_width}{2} \tag{3.9}$$

$$right\_node.y = central\_node.y - sin(\alpha_{radians}) \cdot \frac{lane\_width}{2} \tag{3.10}$$

$$left\_node.x = central\_node.x - cos(\alpha_{radians}) \cdot \frac{lane\_width}{2} \tag{3.11}$$

$$left\_node.y = central\_node.y + sin(\alpha_{radians}) \cdot \frac{lane\_width}{2} \tag{3.12}$$

3.1.3. Monitored area

The concept of monitored area is related to the field of view where a driver should be paying attention. In case of humans this is something instinctive, although not always correct as was demonstrated in [79], but not in the case of self driving vehicles. A first approach could be to analyze every reachable area by the sensors, but this would result in a complex task detecting which elements are affecting the route and which not. Also processing such amount of data requires a large processing capacity.

This can be solved using the HD map together with the route previously calculated. The monitor module can determine which lanes or any other relevant elements are affecting the current route of the vehicle and send that information to other modules as perception module so it can know where to look for obstacles.

Defining which area is monitored is not an easy decision. In this approach we have decided to include every element that interferes with the route in a close distance:

- **Lanes**: current, back, right and left
- **Intersections**
- **Regulatory elements**

In Figure 3.2 are shown the monitored elements along the route from the ego vehicle to a goal point.

The distance to be considered for the current lane is calculated using a braking distance model 3.1.3.1, and it depends on the current velocity of the vehicle. Right and left lanes are only considered if lane change is possible and use the same distance that current lane. Back lane considers a proportional distance to the current lane distance. A threshold is set so the module always consider a minimum distance even if the vehicle is stopped. Intersections and regulatory elements, will be only considered if are affecting the current monitored lane.

3.1.3.1. Braking distance model

The braking distance model returns the distance in front of the vehicle to monitorize depending on its velocity. It calculates a linear regression using two arrays of velocity and braking distance data from [80].

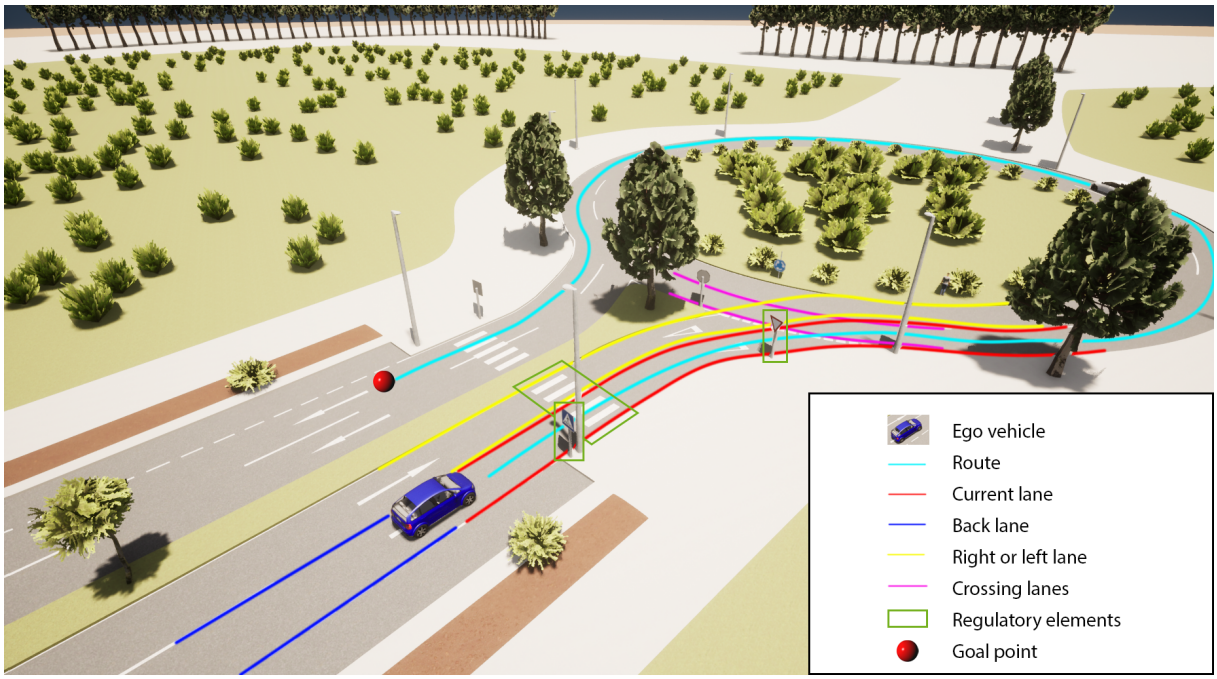
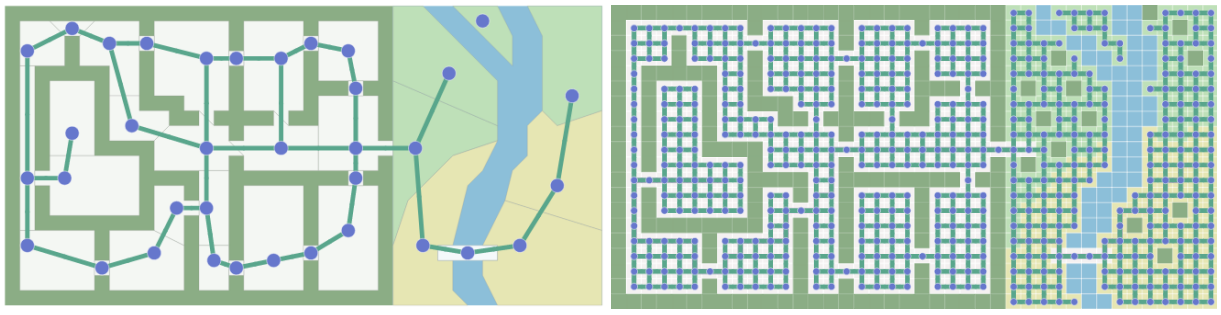


Figure 3.2: Monitored area in CARLA Simulator

3.2. Planning

3.2.1. Graph based path-finding algorithms

We have been talking about different map representations both for global and local planning. One thing to note about the road network is that it is highly structured, which is something we can exploit in our planning process to simplify the problem using a graph. A graph is a discrete structure composed of a set of vertices and a set of edges. For the global planner, each vertex correspond to a given point on the road network, and each edge will correspond to the road segment that connects any two points in the road network. In this sense, a sequence of contiguous edges in the graph corresponds to a path through the road network from one point to another. Using a graph from a topological map has less computational load than other discrete methods like occupancy grid maps, as shown in Figure 3.3.



(a) Topological (b) Ocuppancy Grid

Figure 3.3: Connectivity graphs

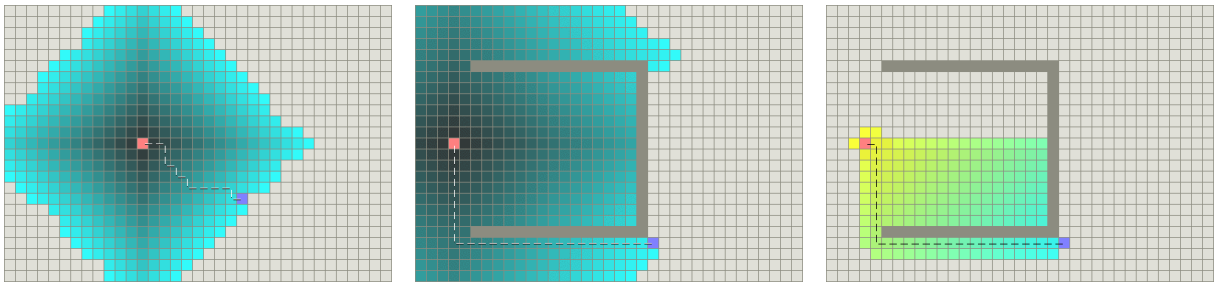
In this section we explain more in detail how some of the most popular graph search algorithms work.

- **Bread First Search**: To find a path using a graph, BFS is the simplest graph search algorithm. It just explores equally in all directions.

- **Dijkstra's Algorithm**: It is also called Uniform Cost Search, and it allows the user prioritize which path to explore. Compared with BFS, it favors lower cost paths instead of exploring equally all possible paths. Costs can be modified by the user to encourage specific moves, like moving on roads instead of off-road paths for example.

- **A***: It is a modification of Dijkstra's Algorithm that has been optimized for reaching a single goal. Dijkstra's Algorithm can find path to multiple locations but A* can only find the path to one location, prioritizing paths closer to the goal destination. For that purpose, both the current distance from the start and the estimated distance to the goal are used.

In Figure 3.4 is shown how each of the algorithms evolve to reach the goal. As expected, A* is optimal when the goal is a single location.



(a) BFS (b) Dijkstra (c) A*

Figure 3.4: Graph based path planning algorithms

Let's see how these algorithms work and why A* can be more convenient over BFS and Dijkstra's. For this explanation, some Python code structures will be used and finally a pseudocode implementation of A* will be presented.

3.2.2. BFS

We build three data structures to aid in our search: open queue of vertices still to be assessed, a closed set of vertices that have been assessed by the search algorithm and a dictionary of predecessors which store the results of the search. A queue is a first-in-first-out data structure, such that the first vertex pushed or added to the queue is the first one popped off or returned from the queue. A dictionary is an unordered set of key-value pairs and for each node in the closed set, stores a predecessor vertex that will identify momentarily.

The algorithm starts by adding our start vertex to the open queue. Then, while the open queue contains vertices, we take the first element from the open queue and check if it is the goal location. If so, we found our shortest path. If not, we then add all adjacent vertices not already in the open queue or closed set to the open queue. This prevents us from getting stuck in cycles during the graph search. Note that by adjacent, we mean all vertices that can be reached from the current vertex. Because we use a queue to store open vertices, we ensure that all adjacent vertices at the current depth in the search are processed before proceeding deeper into the graph. So all vertices that are one step away from the start vertex will be processed before moving on to vertices that are two steps away. As a vertex is added to the open queue, we store its preceding vertex in the predecessor dictionary. This will help us reconstruct the optimal path once the goal is found. Finally, we add the currently active vertex to the closed set and return to the next element of the open queue to process.

3.2.3. Dijkstra's

The overall flow of Dijkstra's algorithm is quite similar to BFS. The main difference is in the order we process the vertices.

The values of each key vertex in the graph will correspond to the distance it takes to reach that vertex, along the shortest path to that vertex we've found so far. In this sense, Dijkstra's algorithm processes vertices with a lower accumulated cost before other ones. A vertex that was added later in the search can be processed before than other that was added earlier, so long as its accumulated cost is lower.

As before, we keep track of the vertices we have already processed in the closed set, and the vertices we've discovered but not yet processed in the open set. The key difference is that instead of using a queue for the open set, we'll be using a min heap. A min heap is a data structure that stores keys and values, and sorts the keys in terms of their associated values from smallest to largest. In our case, the values of each key vertex in the graph will correspond to the distance it takes to reach that vertex, along the shortest path to that vertex we've found so far. In this sense, Dijkstra's algorithm processes vertices with a lower accumulated cost before other ones. Thus, unlike BFS, a vertex that was added later in the search can be processed before when that was added earlier, so long as it's accumulated cost is lower. Other than that, Dijkstra's algorithm is largely the same as BFS. Progressing through the vertices, while adding and popping them off of the min heap until the goal vertex is processed. One interesting case however, is if we find a new path to a vertex that is already in the open heap but has not been processed yet. In this case, we have to check if the newly found path to this vertex is cheaper than the old path. If it is, we need to update its cost in the min-heap. Otherwise, no action is required. Once we process the goal vertex we must have necessarily processed all possible predecessor vertices of the goal node since a predecessor must have accumulated distance less than or equal to the goal vertex. Since all predecessor vertices have been processed, we will have found the shortest path to the goal vertex.

3.2.4. A*

Dijkstra's algorithm requires to search almost all of the edges present in the graph, even though only a few of them were actually useful for constructing the optimal path. This is not a problem for a small example graph, but it will cause issues when we scale our problem to more realistic proportions, such as a full road network for a city.

To improve our efficiency in practice, we can instead rely on a search heuristic by using the A* algorithm to find our destination rather than Dijkstra's.

In this context, a search heuristic is an estimation of the remaining cost to reach the destination vertex from any given vertex in the graph.

Of course, any heuristic we use won't be exact as that would require knowing the answer to our search problem already. Instead, we rely on the structure of the problem instance to develop a reasonable estimation that is faster to compute.

The vertices in the graph correspond to points in space, with the edges corresponding to segments of a road which have a weight corresponding to the length of those road segments.

Therefore, a useful estimate on the cost or length between any two vertices is the straight line or Euclidean distance between them, for a given vertex and goal.

The main difference between Dijkstra's algorithm and A* is that instead of using the accumulated cost, we use the accumulated cost plus hv, the heuristic estimated remaining cost to the goal vertex as the value we push onto the min heap. The min heap then essentially sorts the open vertices by the

estimated total cost to the goal. In this sense, A* biases the search towards vertices that are likely to be part of the optimal path according to our search heuristic. Since we are storing a heuristic based total cost and the min-heap, we also need to keep track of the true cost of each vertex as well, which we store in the cost structure. An interesting thing to note is that if we take our heuristic to be zero for all vertices, which is still an admissible heuristic, we then end up with Dijkstra's algorithm. As before with Dijkstra's algorithm, we will add the origin to the min heap, then pop each vertex off of the heap and add all adjacent vertices to the heap, until we process the goal vertex. A* will never do worse than Dijkstra's, and in practice A* will result in a much faster global planning process.

Listing 3.1 is an example of how to implement an A* algorithm using Python.

Listing 3.1: A* Algorithm Pseudocode

```
1  open ← MinHeap()
2  closed ← Set()
3  predecessors ← Dict()
4  open.push(s,0)
5  while !open.isEmpty() do
6          u, uCost ← open.pop()
7          if isGoal(u) then
8                  return extractPath(u,predecessors)
9          for all v ∈ u.successors()
10                 if v ∈ closed then
11                         continue
12                 uvCost ← edgeCost(G,u,v)
13                 if uCost + uvCost + h(v) < open[v] then
14                         open[v] ← uCost + uvCost + h(v)
15                         costs[v] ← uCost + uvCost
16                         predecessors[v] ← u
17                 else:
18                         open.push(v,uCost + uvCost)
19                         costs[v] ← uCost + uvCost
20                         predecessors[v] ← u
21                 closed.add(u)
```

Chapter 4

Software and Technologies

In this chapter the main software and technologies used in this project will be presented, specifying the version used when needed. It does not pretend to be used as an user guide or a tutorial, so for further information is recommended to consult the official documentation of each technology.

4.1. Linux

Linux is a popular OS that has been growing since it appeared in 1991, started by the student Linus Torvalds in Helsinki.

The Linux open source distribution used in this project is Ubuntu 18.04.5.

4.2. Python

Python is an open source coding language that has become popular in last years.

Python is known because is a legible interpreted language that allows to learn and code faster than other languages.

The version of python used in this project is Python 2.7.

4.3. VSCode

VSCode is the code editor used in this project. It allows to use a Python extension, easing the task of coding and debugging software.

4.4. ROS

ROS (Robot Operating System) is a framework for robotics that provides libraries and tools to ease the development of robotic systems. It is over the open source license BSD and needs to operate over another OS, in this case Linux.

ROS allows to work with multiple modules distributed in different computers. The modules interact with each other through ROS messages, making it possible to work in different coding languages.

The key points of ROS philosophy can be summarized as:

- Peer to peer
- Tools-based
- Multi-lingual
- Thin
- Free and Open-Source

The ROS distribution used in this work is ROS Melodic.

For more information about ROS and how to work with it there is the official documentation [81].

4.5. RVIZ

RVIZ is a 3D visualization tool for ROS applications. In our case, it provides a view of the ego-vehicle model and the data captured from the ego-vehicle sensors.

RVIZ interface can be customized to show only some specific ROS topics in a particular way. Figure 4.1 shows an example of the RVIZ interface. At the left side there are the ROS topics that can be represented with the topic name or the graphical output. The interface is costumizable.

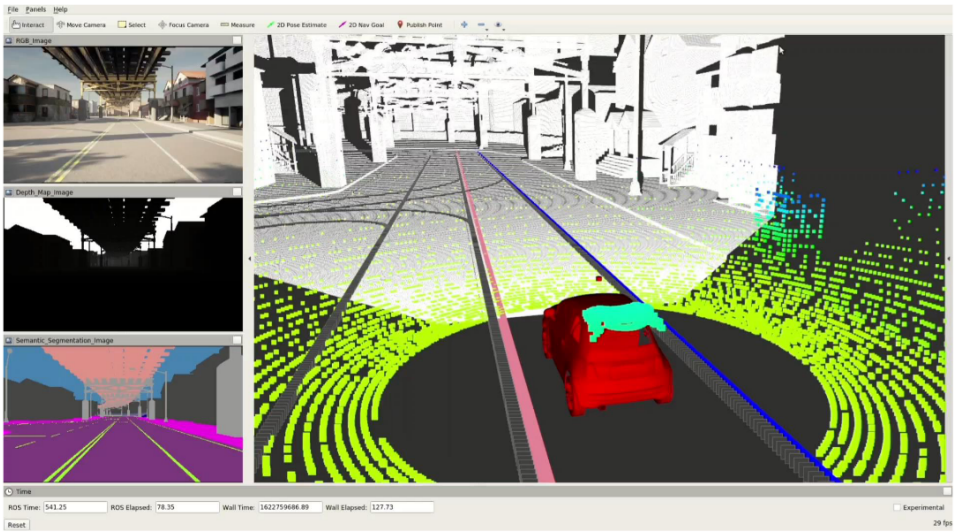


Figure 4.1: RVIZ interface example

4.6. Docker

Docker is an open source project that introduces the concept of containers. The same way a virtual machine virtualizes a server hardware, docker containers virtualize the server OS.

Docker makes development efficient and predictable, accelerating how you build, share and run modern software applications.

For more information about docker, we recommend to consult the official documentation [82].

4.7. Git

Git is a distributed version control system created by Linus Torvalds. It has been used in this project together with GitHub and our own server to keep a clean version control of the code, easing the task of going back to older versions of some code and keeping an updated code workspace in different computers.

4.8. OpenDRIVE

OpenDRIVE is a standard to describe road networks and other relevant road map elements with extensible markup language (XML) syntax, using the file extension XODR. This is what we consider an HD Map 1.2.1.2, and this format in particular is the one that has been used in this project.

OpenDRIVE was originally developed by VIRES and released with the version 1.4. OpenDRIVE 1.4 is the version used in this project, as well as it is the version used when generating maps with RoadRunner 1.2.1.3.

OpenDRIVE standard was recently acquired by ASAM, and latest version has been released under the name of ASAM OpenDRIVE 1.6.

In this section we will survey the main parts to understand how this standard works, for further information we recommend the official documentation [83].

4.8.1. General architecture

OpenDRIVE file architecture is structured in objects with properties that can contain other objects.

The upper level objects are:

- **Header**: This element contains file metadata about the author, the version of the format, date of creation or the reference latitude/longitude for the UTM coordinates.
- **Roads**: After the header, there are the Road elements defining each road of the map. Each road defines its geometry, connections and other elements inside it as lanes and regulatory elements.
- **Junctions**: After roads, there are the Junction elements. A junction is used to solve the connection where more than 2 roads meet the same connection point.

ASAM OpenDRIVE is part of the ASAM simulation standards and is ready to work with OpenCRG and OpenSCENARIO in order to get a scenario description with dynamic and static content (Figure 4.2).

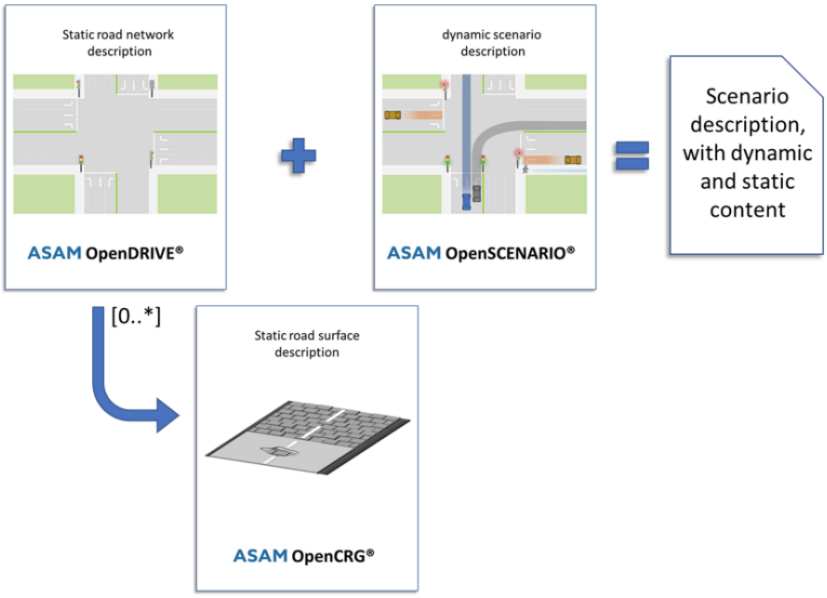


Figure 4.2: Relation between OpenDRIVE, OpenCRG and OpenSCENARIO

4.8.1.1. Road

Roads are the main elements to define an XODR map. A road is based on a reference line defined by a sequence of its geometries that can be lines or arcs, within the lanes and other elements associated with the road (Figure 4.3).

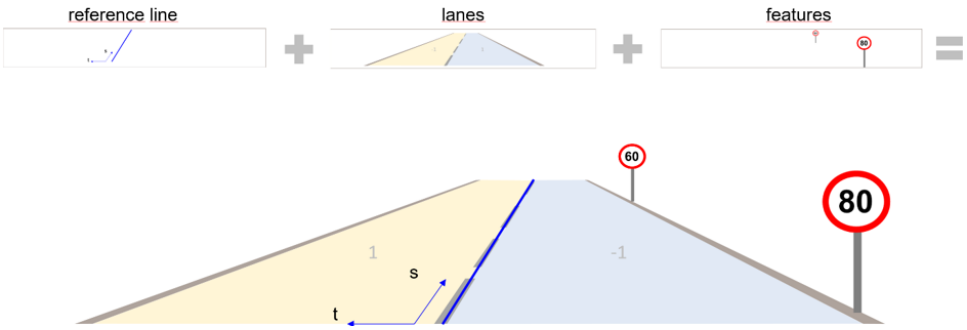


Figure 4.3: Parts of a road

Each road is composed by an unique name, total length of the reference line, an unique id and a flag that indicates if is or not inside a junction.

The different XML structured elements in a road are:

- **link**: Link indicates predecessor and successor elements, their unique ids and their type than can be Road or Junction.
- **type**: Type of road, that in our case is always Town. There is also a parameter with the maximum speed of the road.
- **planView**: This is the key parameter to locate roads in the UTM XY plane. The PlanView is composed by a sequence of geometries, that can be lines or arcs, defining the shape of the road. Each geometry has the parameters:

- **s**: Distance in the reference line where this geometry starts. The first is always 0 and the last the total length of the road minus the length of the last geometry.
- **x**, **y**: x, y UTM coordinates where the s parameter of the geometry starts.
- **hdg**: Heading in radians of the origin of the geometry.
- **length**: Total length of the geometry. The sum of all geometries must be equal to the total length of the road defined in the first parameters of the road.

- **elevationProfile**: Defines the elevation profile (Z) for each segment of the road indicating the s parameter where it starts in ascending order along the reference line. The elevation profile is calculated using the coefficients with the following polynomial function of third order:

$$elev(ds) = a + b \cdot ds + c \cdot ds^2 + d \cdot ds^3 \tag{4.1}$$

  Where a, b and c are parameters defined in the elevationProfile element and ds is the distance along the reference line.

  The length does not change with the elevation.

- **lateralProfile**: elevationProfile specifies the elevation along the road reference line, that is in s direction. The lateral profile specifies the elevation orthogonally to the reference line, that is in t direction (Figure 4.4).

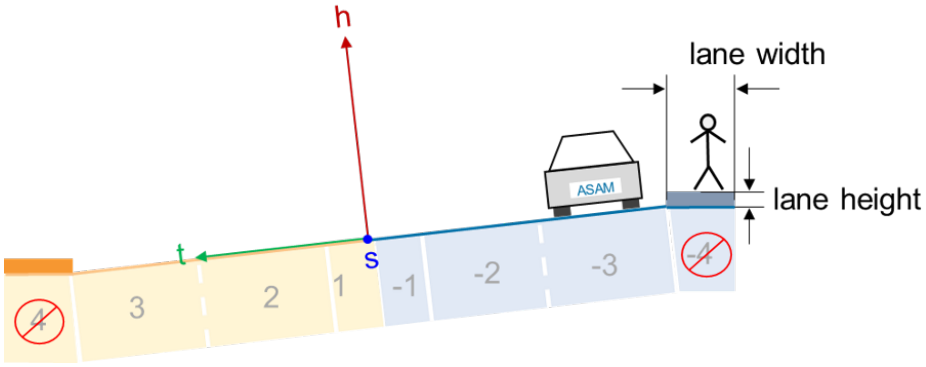


Figure 4.4: Elevation and lateral profile in a road

- **lanes**: Lanes are complex elements and will be defined in more detail in 4.8.1.2
- **objects**: Objects are items that influence a road by expanding, delimiting, and supplementing its course. The most common examples are parking spaces, crosswalks, and traffic barriers. Every object has a name and an unique id, and are referenced to the reference line by its s, t, heading and orientation.
- **signals**: Signals are traffic signs, traffic lights, and specific road marking for the control and regulation of road traffic (Figure 4.5).

  Each signal has a name and an unique id, and is referenced to the reference line similar to object elements. Signals also have a laneValidity parameter to indicate the lanes affected by the signal.

4.8.1.2. Lane

In OpenDRIVE, all roads contain lanes. Each road shall have at least one lane with a width larger than 0. The number of lanes per road is not limited. Lanes element has laneOffset and laneSection

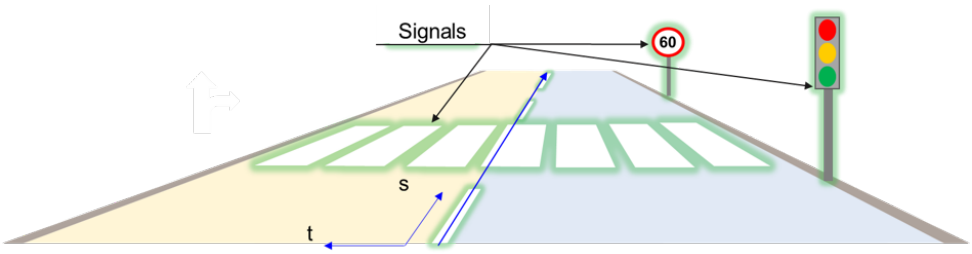


Figure 4.5: Signals in OpenDRIVE

parameters.

LaneOffset is used to define an offset regarding the reference line where the lanes may start to be defined.

LaneSection is used to differentiate parts of the road with a different number of lanes (Figure 4.6).

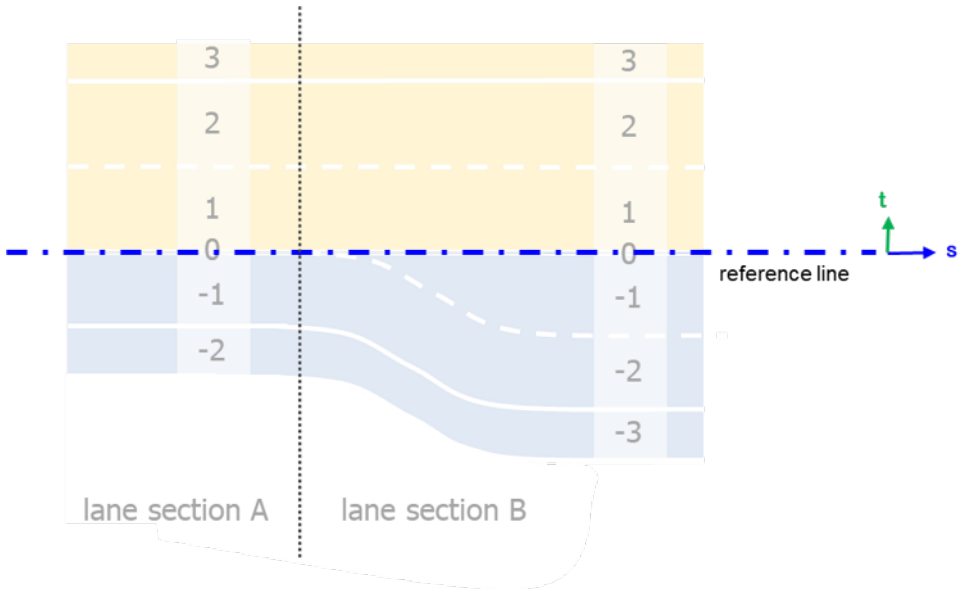
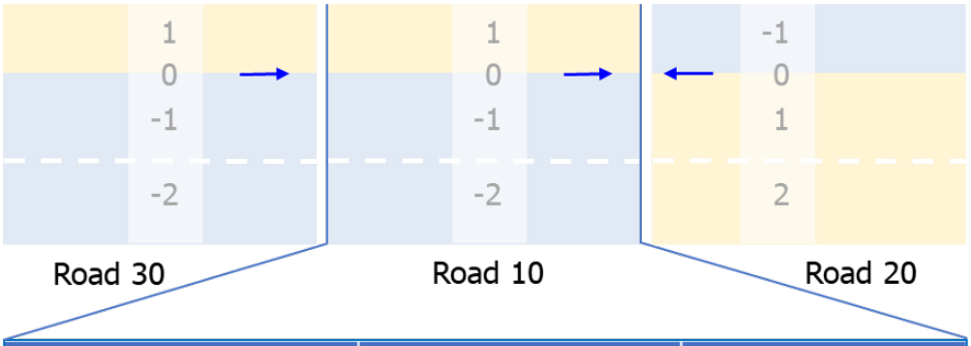


Figure 4.6: A road with lane sections

In each lane section are considered three different parts: center, right and left lanes, and for the three of them the lanes are defined using the same parameters. Right and left lanes are considered respect to the origin of the reference line, that may not be equal to the driving direction of the road. Right lanes have negative id values and left lanes have positive id values.

Every lane has an unique id, the type, that can be driving, shoulder or border among others, and these other parameters:

- **link**: Link indicates the predecessor and successor lane by its id. Depending on how the reference line of the road is defined, lane id number may not match (Figure 4.7).
- **width**: The width of a lane is defined along the t-coordinate. The width of a lane may change along a lane section.
- **roadMark**: Road lane marks can be defined using different colors and styles. This parameter also defines if lane change is possible or not. Lane change can be left, right, both or none, depending on the lane mark.



Road / Lane	Lane predecessor	Lane successor
Road 10 / lane id 1	Lane id 1	Lane id -1
Road 10 / lane id 0	Lane id 0	Lane id 0
Road 10 / lane id -1	Lane id -1	Lane id 1
Road 10 / lane id -2	Lane id -2	Lane id 2

Figure 4.7: Lane links for road with id 10

- **speed**: This parameter can be used to specify the maximum speed allowed of each lane. Lane speeds limits overrides road speed limits.

There is an extra parameter that is worth mentioning: userData. userData contains custom user data that is not yet described in the standard OpenDRIVE. Examples are different road textures.

4.8.1.3. Junction

Junctions are areas where more than 2 roads meet. Each junction has an unique id and name. Each junction can have multiple connections that are formed by 2 types of roads: incomingRoad and connectingRoad (Figure 4.8). IncomingRoads are the roads containing lanes that lead into a junction and connectionRoads represent the paths trough a junction. For each connection is also defined from which lane to which lane is the connection path.

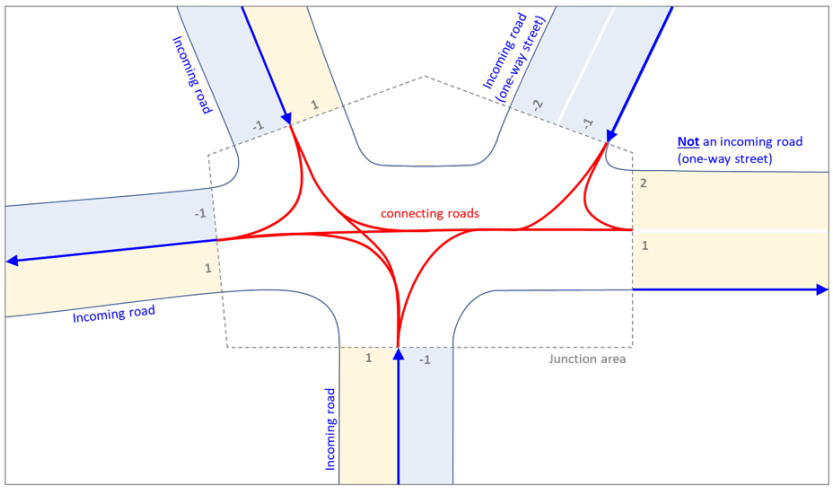


Figure 4.8: Junction in OpenDRIVE

4.9. Carla Simulator

Carla Simulator [13] is an ongoing open-source hyper-realistic simulator (Figure 4.9) designed up to support development of autonomous driving systems. The simulator is based on Unreal Engine [84] and provides open digital assets (urban layouts, buildings vehicles) that were created for this purpose and can be freely used.

Figure 4.9: CARLA Simulator

The simulator presents multiple advantages:

- **Autonomous Driving sensor suite**: There are multiple sensors configured to be integrated in the simulation such as LIDARs, cameras, depth sensors and GPS, among others.
- **Documentation**: Carla Simulator has published an extended documentation to clarify how to set up the simulator environment and how to use the platform. They have three main sites to solve every doubt:
  - **CarlaReadTheDocs**: This is the official web site of the Carla documentation. There is explained how to install the simulator, first steps, advanced steps and many other options that are included.
  - **GitHub**: There is also documentation on their GitHub public account and doubts can be solved using issues option.
  - **Discord**: Finally, they also have a private server in the application Discord. This is the most pragmatic way to solve doubts interacting with the developers.
- **Python API**: A PythonAPI is released within the simulator to ease the use of the simulator features. The API is explained in detail on their documentation, in some cases even using code examples.
- **Map generation**: Some OpenDRIVE maps are released within the installation, but new maps can be designed using applications like RoadRunner 1.2.1.3.
- **ROS integration**: ROS integration is provided using a ROS-bridge.

- **Traffic scenarios simulation**: The module ScenarioRunner allows traffic scenario definition and execution for CARLA Simulator.

The version of Carla simulator used in this project is **CARLA 0.9.10**.

4.10. RoadRunner

RoadRunner [7] is an interactive editor that allows to create 3D scenes for simulation and testing of self-driving systems. It has different tools and options for creating any traffic element like traffic signs, traffic lights, crosswalks, among others. Also miscellaneous elements like bushes or threes can be added to create a more realistic environment. RoadRunner provides tools to synchronize traffic lights and driving paths in intersections.

This software supports the visualization of data from LiDAR point clouds, aerial images and GIS. It allows to import and export road networks with OpenDRIVE. The 3D scenes generated using RoadRunner can be exported using different formats: FBX, gITF, OpenFlight, OpenSceneGraph, OBJ and USD. The exported scenes can be used in self-driving simulators and game engines, including CARLA, Vires VTD, NVIDIA DRIVE Sim, LGSVL, Baidu Apollo, Unity and Unreal Engine.

In our case we will be exporting the scenes generated in OpenDRIVE (.XODR) and FBX formats for using the maps in Carla Simulator, that is powered by Unreal Engine.

Figure 4.10 is an example of how the interface of RoadRunner looks.

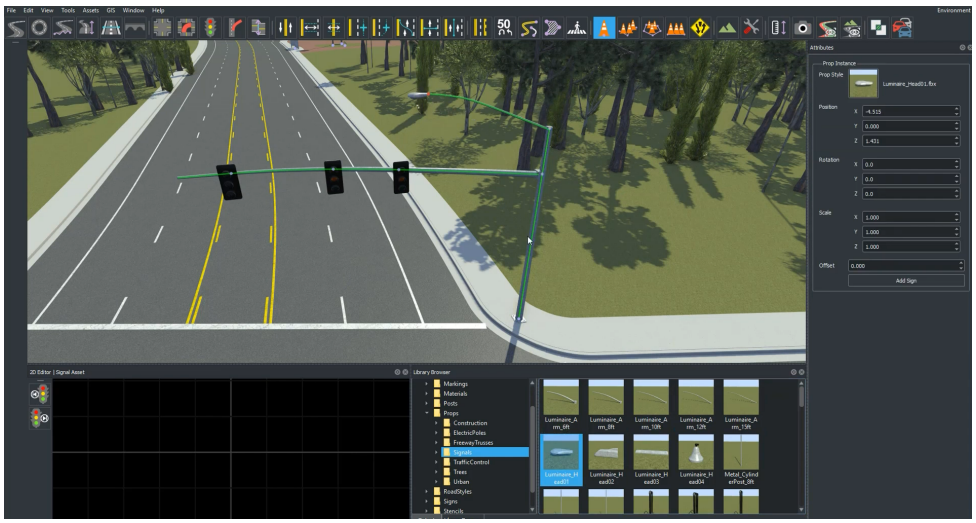


Figure 4.10: RoadRunner interface

In Figure 4.11 is shown how complex junctions with traffic sign can be modeled.

Figure 4.12 is another example of how custom traffic signs can be created.

Another interesting feature used in this project is the importation of GIS images (Figure 4.13) for creating the map with precise locations.

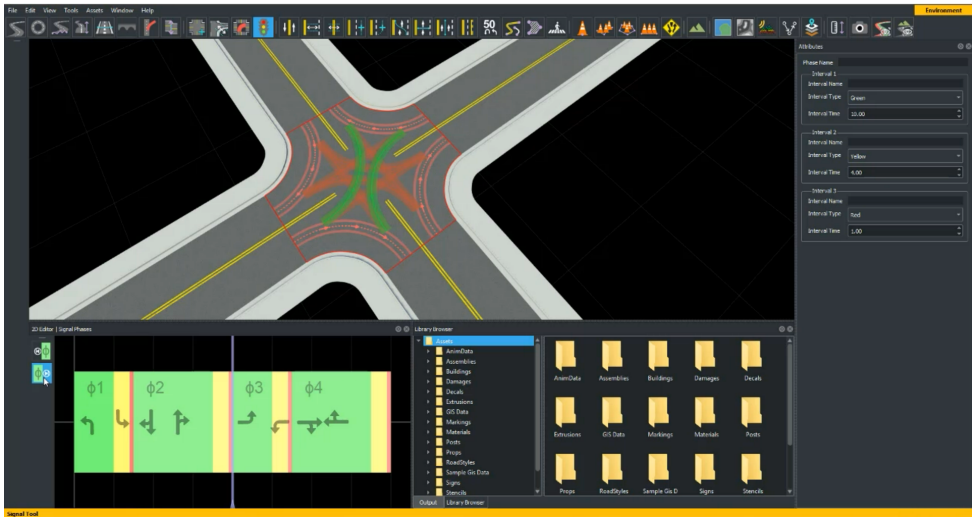


Figure 4.11: RoadRunner complex junction example

Figure 4.12: RoadRunner custom traffic sign example

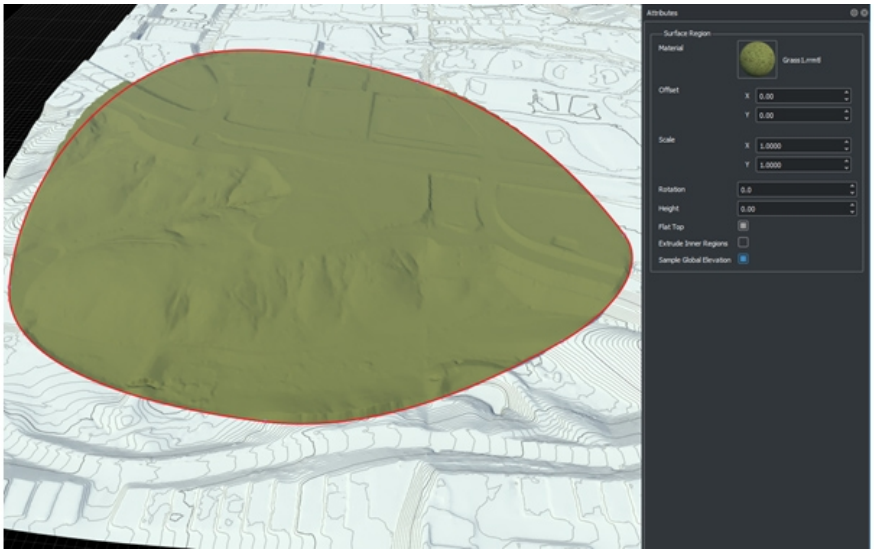


Figure 4.13: RoadRunner GIS image

Chapter 5

Development

This chapter aims to explain the modules developed in this work, referencing code slices when needed.

First of all is the map generation. Then, the different code modules will be explained.

The first module is the Map Parser. It is the module in charge of getting the information from the HD Map and organize it into classes, turning it into useful information for the other modules.

After that, the Map Monitor will be explained. This module uses the information parsed from the HD Map to monitor the surrounding elements to the vehicle that are affecting the route.

Then, the Path Planner, using the information parsed from the HD Map too, builds a road-lane directed graph and applies an A* algorithm to generate a waypoints path.

5.1. Map Generation

The map generation is done using the RoadRunner tool. This part includes all the process from the data acquisition using sensors in the real world until the final map file that is used in the simulator and by the mapping and planning modules.

We can summarize the process in three main parts:

- Data acquisition
- Map generation in Roadrunner
- Map import in Carla Simulator

Data acquisition The first step is to map discrete points of the road boundaries, walking on the road sides and using a DGNNS (Differential GPS). This data is saved in a .gpx file, keeping a record of the geographical points.

Map generation in Roadrunner The .gpx file is imported in RoadRunner, so there is an accurate reference blueprint to define the roads with precision. When the .gpx is imported, a reference point is set so the map is georeferenced and UTM system can be used.

A satelital image is also imported so it can be used as a reference for creating the environment and adding other scene elements in addition to the roads, but the roads are defined following the .gpx reference because the satelital image is not accurate enough.

Map import in Carla Simulator Once the map is finished, the Roadrunner is exported so it can be used in Carla Simulator and in the project. The files exported are:

- .fbx
- .xml
- .xodr

These three files are can be imported in Unreal Engine so the map can be used in Carla Simulator.

Moreover, only the .xodr file is needed to be used by the mapping and planning layer. It is a text file defining the road map that can be parsed and used for an offline map monitoring and path planning.

Figure 5.1 illustrates the steps of map generation.

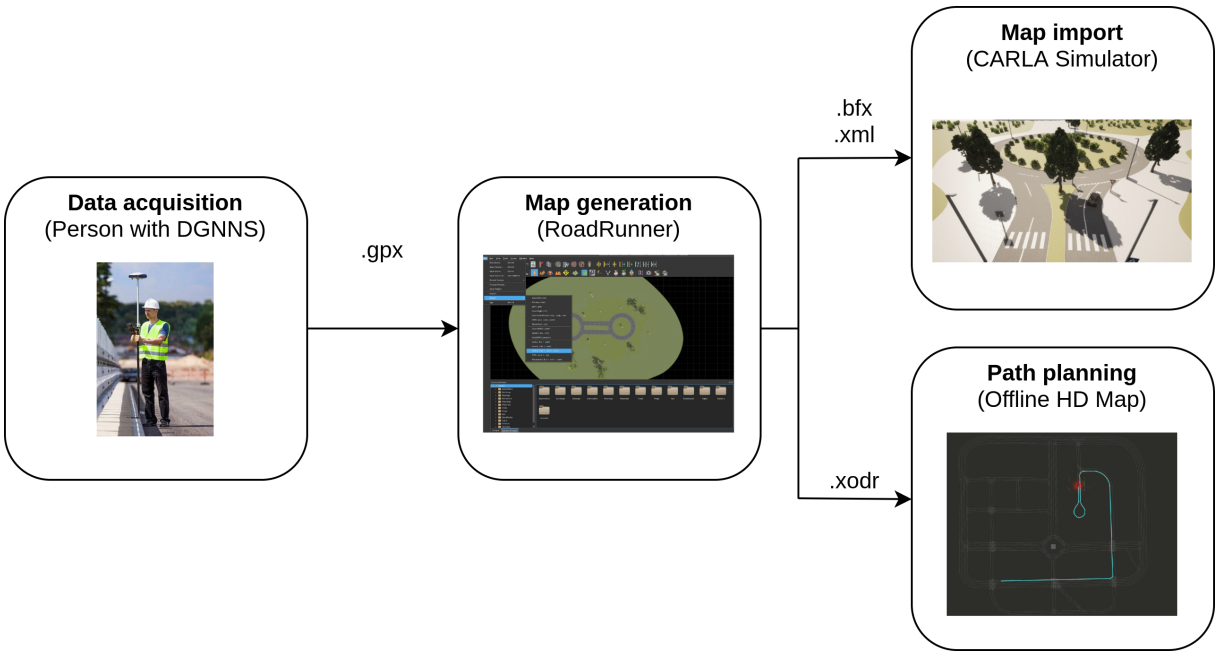


Figure 5.1: Map generation steps

5.1.1. Why ASAM OpenDRIVE?

In this project we have not tried to create a new HD map format to satisfy our requirements, because this would be contributing to avoid a standardization of the HD maps. Instead, we have studied the different existing formats to choose the one more appropriated to our project: OpenDRIVE.

As is shown in TABLE 1 of [8], Figure 5.2 in this document, OpenDRIVE gets the highest score in most of the cases, having also other advantages that we will see next.

By the other hand, it also gets the worst score in the complexity of the format. This is because it offers a huge quantity of metadata, but in exchange the structure of the format is not that simple. That means more difficulties when generating and processing the map.

If we compare to OSM, this format is made up of tagged cartograic points. It does not have any road structure and this is why needs to be complemented with an extra framework as Lanelet. In the case of XODR is more complex because it already have a road map structure implicit in the format.

Models \ Features	Primitive smoothness	Lane Metadata	Topological information	Coverage/ Generation complexity	Travel data independence	Database Query	Centreline available (for path planning)	Margins independent from centreline
RDDF [16]	0	+	0	++	+++	0	0	0
RNDF [17]	0	++	+	++	+++	0	0	0
RNDGraph	++	++	+	++	+++	0	0	0
OSM [8]	0	++	++	+++	+++	0	0	0
OpenDrive [18]	+++	+++	+++	+	+++	0	++	++
Lanelet [9]	+	+++	++	+++	+++	0	+	+
Extended Maps [19]	++	+++	++	++	+++	0	+	++
Akima intrepolation [17]	++	+	+	+	0	0	++	++
Enhanced maps [10]	+++	+	+	+	0	0	+++	+++
ADASIS [5]	+	+++	+++	++	+++	++	+++	0
Self-generated corridors	+++	++	++	+++	+++	+	+++	++

Figure 5.2: Overview of the main features for the different road geometry mapping models [8]

However, complexity problem when generating maps can be solved using specific software, as in our case RoadRunner.

The other problem related to complexity when processing data can also be solved using an specific library that simplifies the process. In our case we have used the existing library libCarla, developed by Carla Simulator. For other functionalities that are not in this library we have developed our own modules, trying to contribute to the idea of creating a standard open source library to manage XODR maps. This way future users may not have to deal with the worst characteristic of ASAM OpenDRIVE, its complexity.

Another disadvantage, shared with most of the other formats, is that there is not any source of XODR maps so if you want to use an specific map you have to create it first.

The last reason why have we chosen this format, is that in our project we work with Carla Simulator. Native format for the simulator maps are XODR, so in our case we are working with exactly the same maps in the real vehicle and in the simulator. This way the algorithms can work exactly the same in both real and simulation situations.

5.2. Map Parser

The Map Parser module is in charge of getting the information of the map from the XODR file and transform it into custom classes that can be used by other modules.

In consists in two files (Figure 5.3):

- map_classes.py
- map_parser.py

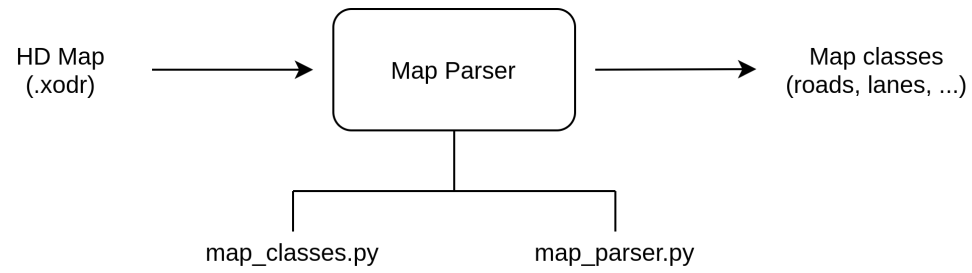


Figure 5.3: Map Parser structure

5.2.1. map_classes.py

To store the information from the xodr file, some custom classes have been created to reproduce the roadmap structure of OpenDRIVE. Not all the information is stored, but only the useful for our project.

Some of the classes contain other classes, in addition to its own parameters. The classes and its parameters are structured under 3 main classes: T4ac_Header (Figure 5.4), T4ac_Road (Figures 5.5 5.6 5.7) and T4ac_Junction (Figure 5.8).

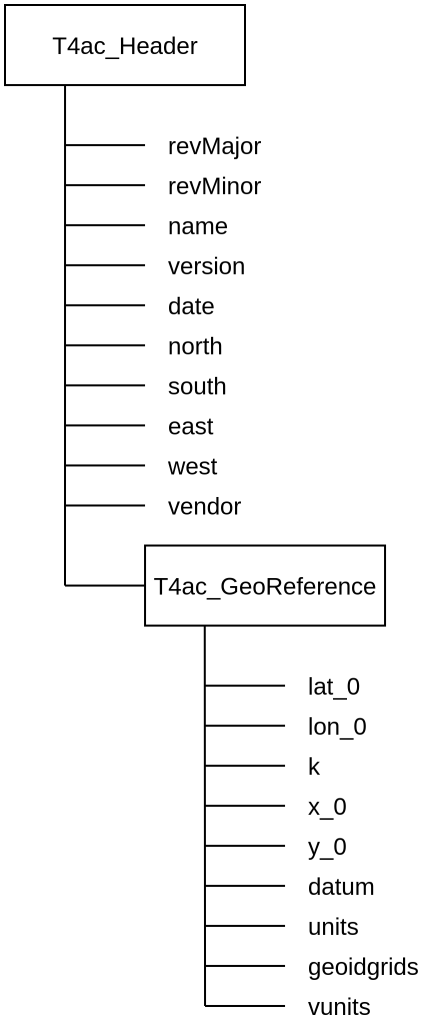


Figure 5.4: T4ac_Header class for Map Parser

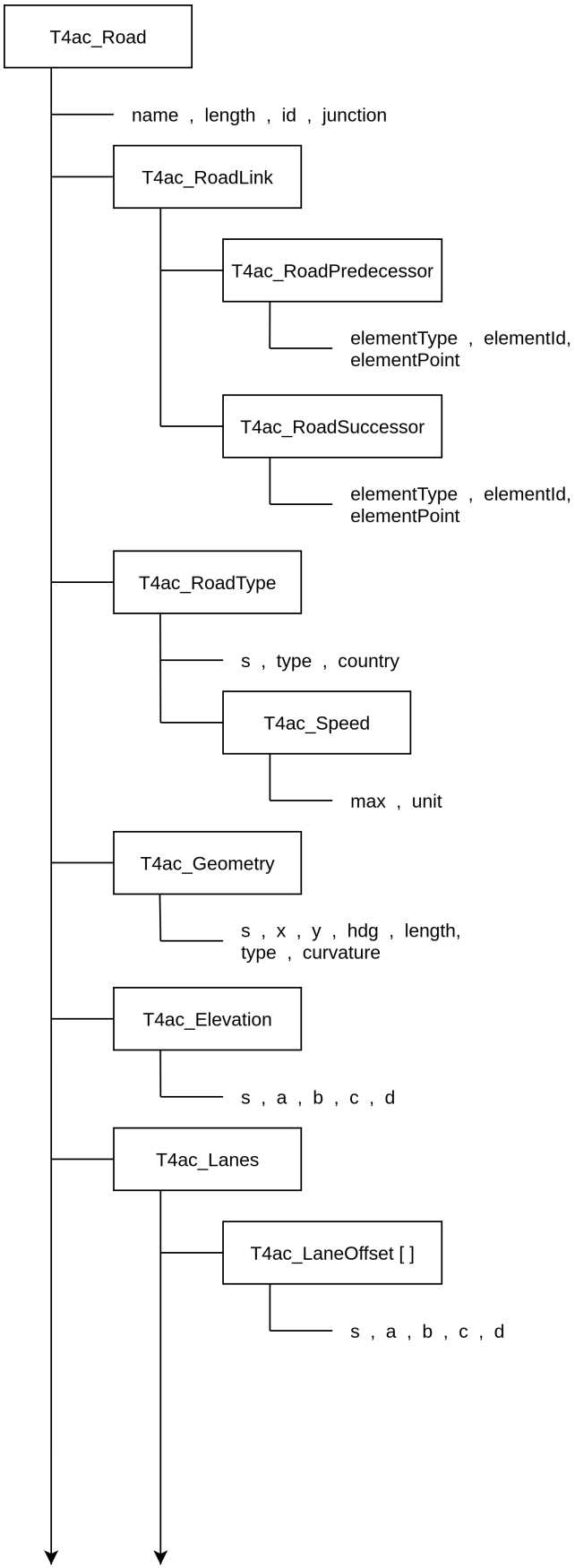


Figure 5.5: T4ac_Road class for Map Parser. Part 1

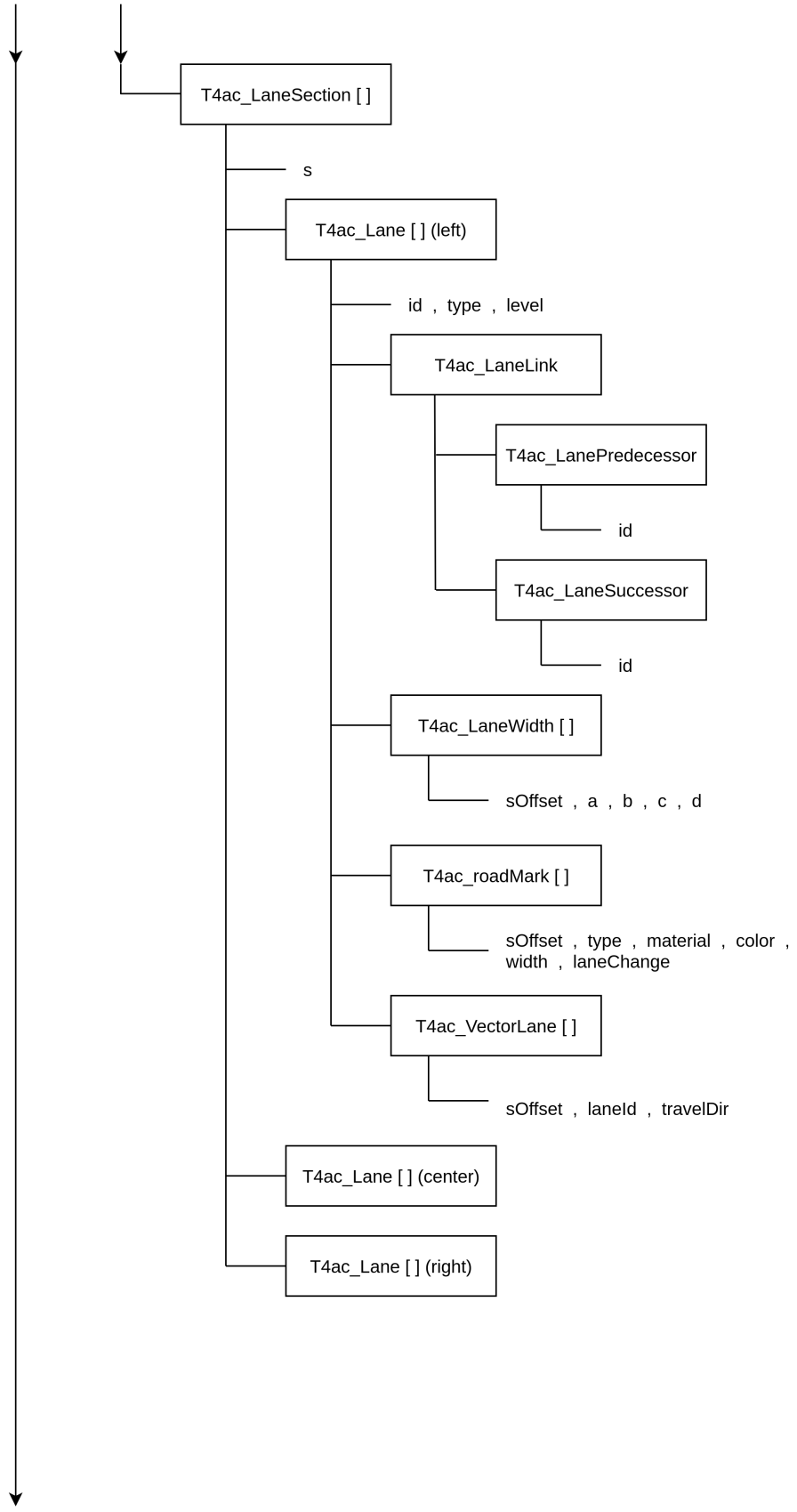


Figure 5.6: T4ac_Road class for Map Parser. Part 2

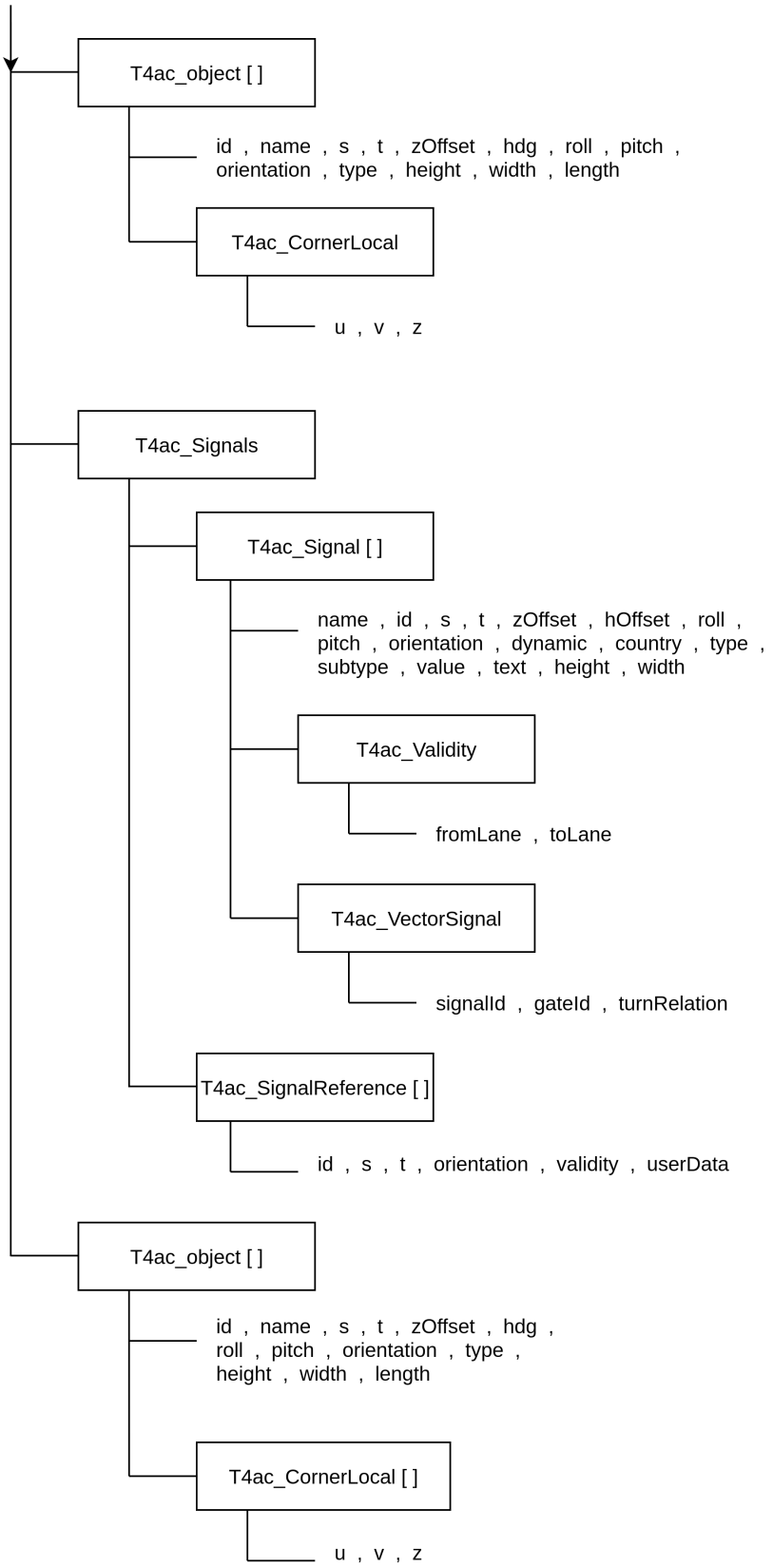


Figure 5.7: T4ac_Road class for Map Parser. Part 3

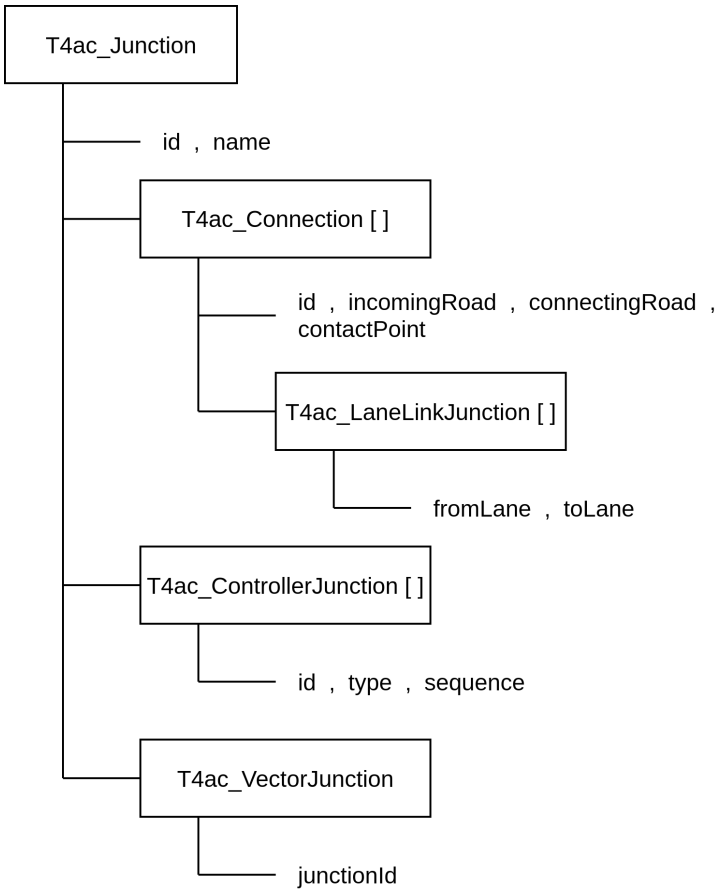


Figure 5.8: T4ac_Junction class for Map Parser

5.2.2. map_parser.py

This module receives the XODR file and analyze every single text searching for characters matching the classes names, detecting the objects and storing them into the map_classes previously defined. The only input is the XODR file, and the output is a map object containing all the map parsed information.

5.3. Map Monitor

The Map Monitor module is the part in charge of monitoring the surrounding area of the vehicle and visualizing both the monitored area and the lanes describing the roads of the map.

To do that, it has 3 main python files and some python modules with useful functions implemented (Figure 5.9).

- **map_monitor.py** : Calculate the monitor elements and publish them into ROS topics.
- **monitor_visualizator.py** : Get the monitored elements from the map_monitor.py and publish lane markers to visualize them in RVIZ.
- **map_visualizator.py** : Calculate the map topology from the HD Map file and publish markers to visualize the map in RVIZ.

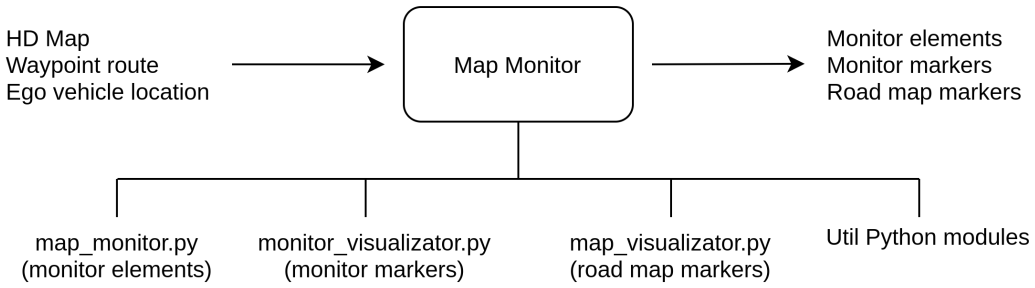


Figure 5.9: Map Monitor structure

The inputs of the Map Monitor are the XODR Map, the waypoint route (to know which area to monitor) and the ego vehicle location to know in which part of the route is the ego vehicle.

The outputs are the monitored elements published into ROS topics using custom messages and markers published into ROS topics too to visualize the elements in RVIZ.

Hereafter we will explain how the code works.

5.3.1. map_monitor.py

This script is subscribed to 2 ROS topics: waypoint route and ego vehicle position. Every time a message is published in any of the topics a callback function starts, so the code is structured in two callback functions:

- route callback
- location callback

Also there is a header part initializing the ROS publishers where the monitor elements will be published once they are obtained.

5.3.1.1. Route Callback

This function is called when a route is published by the path planner.

The route is published as a custom t4ac_msgs/Path.msg ROS message, that consists in a header and a list of t4ac_msgs/Waypoint.msg messages. A waypoint message contains some topology information about the road and a transform with location (x, y, z) and rotation (pitch, yaw, roll).

The callback pseudocode is shown in Listing 5.1.

Listing 5.1: Route Callback Pseudocode

```
1 if ego vehicle is inside the route
2         get in which segment of the route (0 by default)
3         activate monitor
4 else
5         deactivate monitor
```

5.3.1.2. Location Callback

Location callback is called every time that ego vehicle position is published in its ROS topic, that in a real case is constantly published.

The pose of the vehicle is published using a nav_msgs/Odometry.msg.

This callback check some conditions and if every thing is correct then calculates the monitored elements for a given distance and publishes them in their corresponding ROS topics.

The threshold distance to monitor the current lane is obtained using a braking distance model 3.1.3.1.

The monitored elements are:

- **Lanes**: current, back, left and right lanes. Current lane is monitored from current position to a dynamic distance depending on the velocity of the ego vehicle. Left and right lanes are monitored the same distance that current only if lane change is possible. Back lane is monitored from current position to back a proportional distance of the dynamic current lane distance obtained. The lanes are published using a custom ROS message t4ac_msgs/MonitorizedLanes.msg that contains a Header and a list of t4ac_msgs/Lane.msg. Each lane is defined by two t4ac_msgs/Way.msg (a list of xyz nodes) defining its right and left boundaries.
- **Intersections**: Other lanes that intersect the current monitored lane are checked. Intersections can have different roles: split (1 lane splits into 2 or more), merge (2 or more lanes merge into 1) and cross (a lane crosses a part of the current lane). Monitored intersections are published also using custom ROS messages t4ac_msgs/MonitorizedLanes.msg. To calculate the intersection lanes, each lane of every junction in the current lane is evaluated (Remember that junctions are areas where more than 2 roads meet). The polygon of each lane is calculated and evaluated if is inside the polygon of the current lane (Listing 5.2).

Listing 5.2: Monitored Intersections Pseudocode

```
# Get junctions affecting the current lane
junctions_in_current = get_junctions(current_lane)
for junction in junctions:
    # Compare current lane polygon with junction lane polygon
    if first waypoint is out and last waypoint is in → merge
    elif first waypoint is in and last waypoint is out → split
    elif first and last waypoints are out, but some others are in → cross
    else → none
    append element to monitored_intersections
return monitored_intersections
```

- **Regulatory elements**: This is an ongoing part of the work. Currently, it has been developed using the PythonAPI from Carla to get the landmarks corresponding to the close regulatory elements (Listing 5.3). The problem of using the PythonAPI is that not all the regulatory elements in the HD Map are considered, only traffic lights, yield and stop signs, but not the crosswalks for example. As a future work this part must be extended. A landmark is a sign indicating the place in the lane where the regulatory element takes effect. The regulatory elements are published in ROS topics using t4ac_msgs/RegulatoryElement.msg custom message, containing the position of the regulatory element, the position of its corresponding landmarks, the distance from the ego vehicle position, and other useful information.

Listing 5.3: Monitored Regulatory Elements Pseudocode

```
# Get landmarks close to the ego vehicle
close_landmarks = get close landmarks
# Filter landmarks affecting the route
affecting_landmarks = filter landmarks affecting the route
# Get regulatory element objects
for landmark in affecting_landmarks:
        get regulatory element
        append regulatory element to regulatory_elements
return regulatory_elements
```

5.3.2. monitor_visualizator.py

The monitor visualizator is subscribed to the topics published by the map monitor. The information from the map monitor custom ROS messages is converted into ROS markers to visualize the monitored elements in RVIZ.

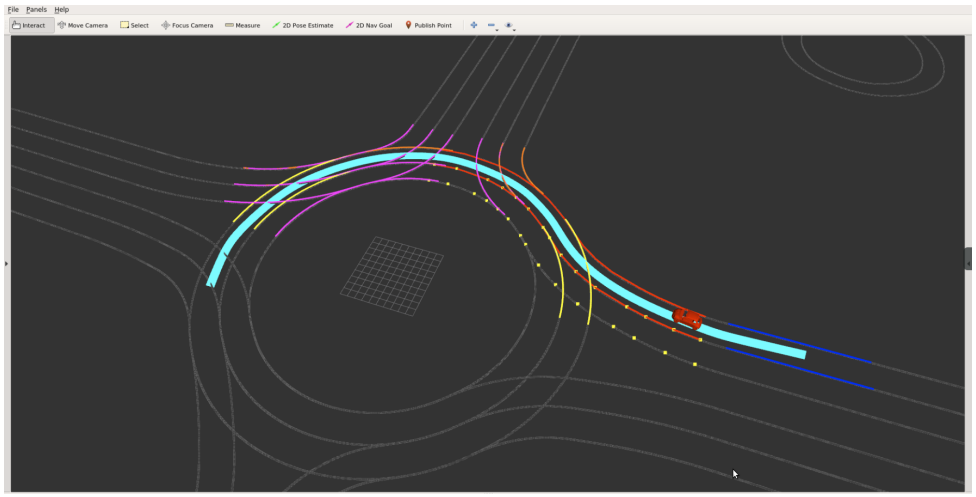
To facilitate visualization understanding we apply the following color code:

- Current lane: red lines
- Back lane: blue lines
- Right and left lanes: yellow points
- Split intersections: orange lines
- Merge intersections: yellow lines
- Cross intersections: purple lines
- Regulatory elements: red points

In Figure 5.10 an example of the map monitor is shown in Town03 of Carla Simulator.

This module has an only visualization purpose. In order to use the monitor information for calculus in other layers as perception, the custom ROS messages published by the map_monitor.py must be used instead.

(a) Carla simulator



(b) RVIZ

Figure 5.10: Map monitor in Town03

5.3.3. map_visualizator.py

This module has a similar purpose than 5.3.2, but in this case the map topology is visualized. The map topology is represented using the lane boundaries of the whole map with ROS markers. The color code to represent the lane boundaries is grey lines (Figure 5.11).

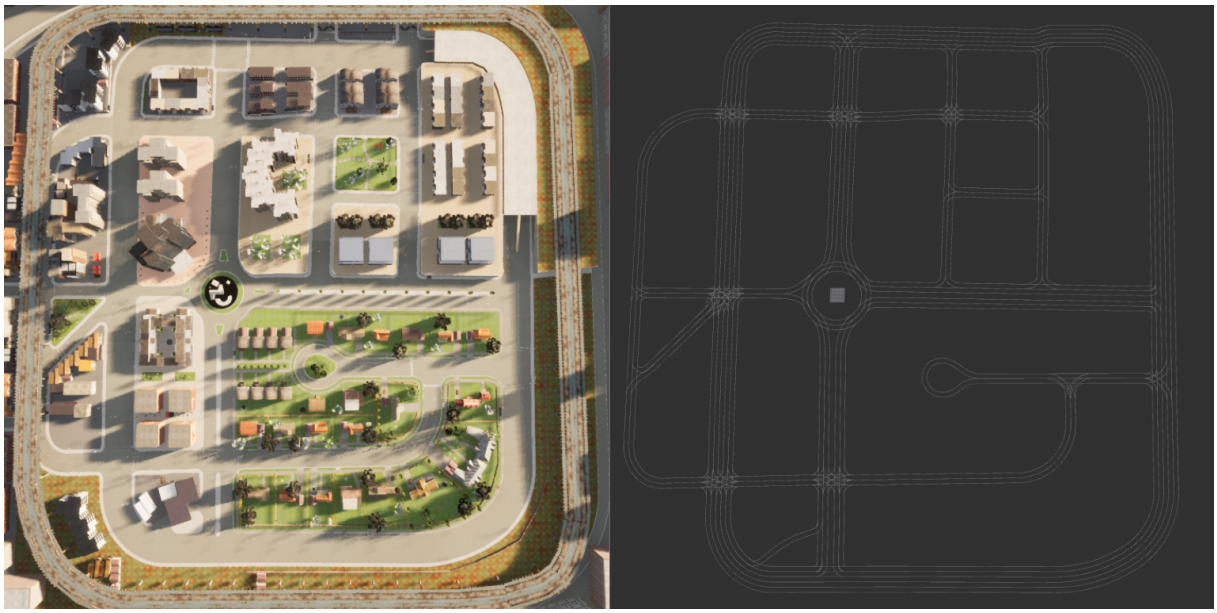


Figure 5.11: Carla Simulator Town03 and its representation in RVIZ by Map Visualizator

The Map Visualizator considers height of the map (Figure 5.12).

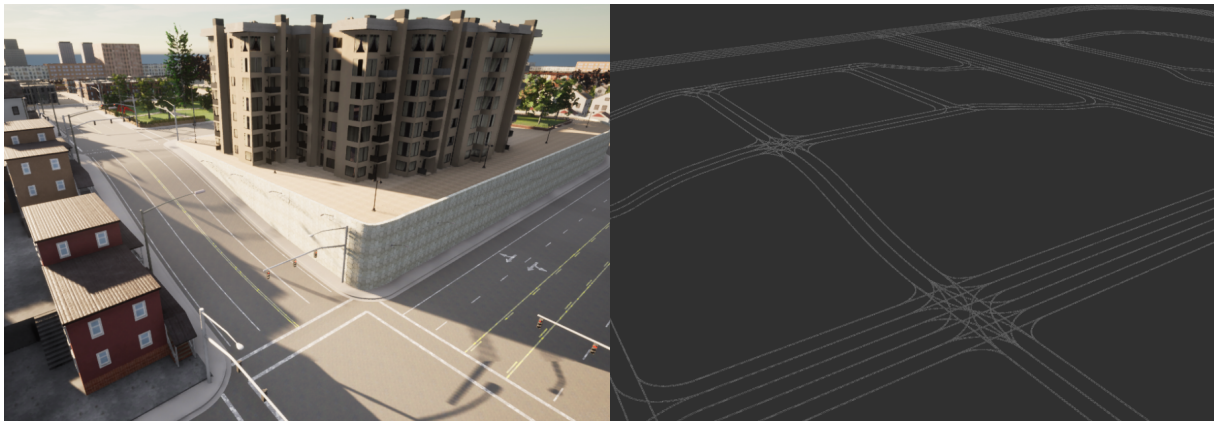


Figure 5.12: Height change in Town03 and its representation in RVIZ by Map Visualizator

5.4. Path Planner

The path planner is the module that receives the ego vehicle position and goal position and calculates the route between them as a list of waypoints centered in the lane.

To do that, it applies an A* algorithm to a road-lanes graph previously generated from the HD Map to obtain a topology (road-lane) route. Then, it generates waypoints centered in every lane of the route separated by a given distance.

It also considers lane change when the order is published by the decision making layer.

It is in a different layer of the project that the map monitor and the map parser. It is in the t4ac_planning_layer, and inside the layer this path planner is a ROS package t4ac_global_planner_ros.

The main parts of the path planner are organized into these python files:

- lane_graph_planner.py 5.4.1

- lane_waypoint_planner.py 5.4.2

- map_utils.py

The inputs and outputs of the path planner are shown in Figure 5.13.

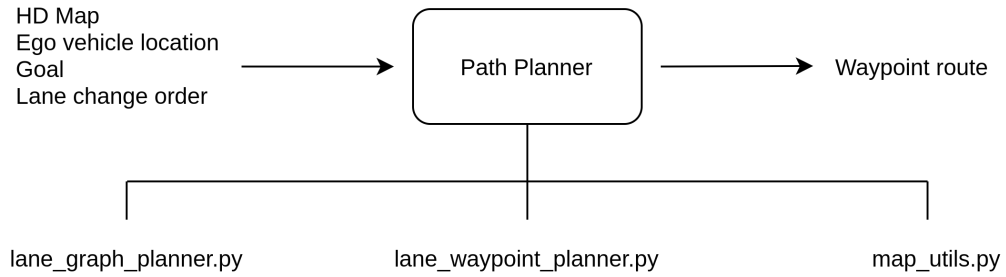


Figure 5.13: Path Planner structure

5.4.1. lane_graph_planner.py

Lane graph planner (LGP) generates a directed graph (DiGraph) of roads and lanes from the HD Map using the python module networkx [85].

The LGP builds the graph using edges, that represent the connection between two nodes. In this case a node is a tuple of 2 parameters: road_id and lane_id. Each edge is defined as a python set containing the input node, output node and weight (Listing 5.4):

Listing 5.4: Graph edge structure

```
1 edge = ([(input_road_id , input_lane_id),
2           (output_road_id , output_lane_id),
3           weight])
```

The weight represents the cost that will be used by the graph planner for going from the input node to the output node. In this case the weight is the distance in meters, but could be changed by any other unit as for example the time for going from the input node to the output at the maximum velocity of the road.

To generate edges and build the graph, the LGP evaluates connections for every road/lane of the map object parsed from the HD Map. A first loop is used to evaluate every road and a second nested loop evaluates every lane connection of each road. Lane change is also considered adding a cost value for lane change when it is allowed.

Figure 5.14 is an example of how the graph of a map with two roundabouts is generated, representing also the distances between nodes.

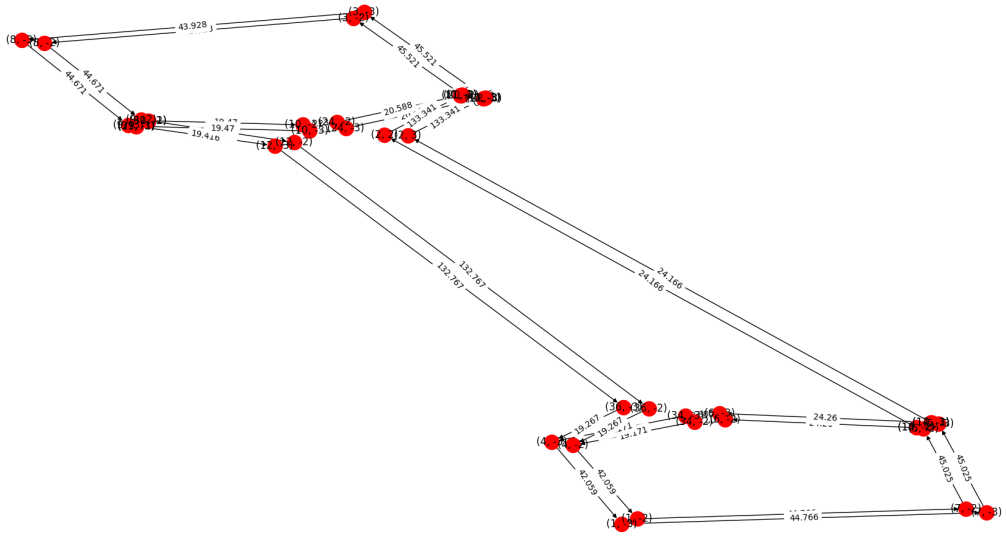


Figure 5.14: Two roundabouts DiGraph using Netwokx

Once we have the road graph, the route is calculated using an A* algorithm already implemented in the Networkx module as a method:

```
1 route = netwokx.astar_path(graph, init_roadlane, goal_roadlane)
```

The route returned by the LGP is a topological route as a list of tuples: (road, lane, action). Action can be lane follow, lane change right or lane change left. It could be used as an input to a perception based controller, but in this case a list of waypoints must be given to the controller. This is generated in 5.4.2.

5.4.2. lane_waypoint_planner.py

The Lane Waypoint Planner (LWP) is in charge of calculating a route of waypoints centered at the lane from an initial location to a goal point.

It is an extension of the LGP, keeping all its functionalities and adding new ones.

The different processes are organized in callback functions that are activated when a message is published in the corresponding ROS topic.

ROS subscribers:

- pose : This topic constantly updates the pose of the ego vehicle
- goal : When a goal is published, a new route is calculated. To do so, first calculates the topology road-lane-action route described in the LGP and then obtains waypoints centered at every lane of the route separated by a given distance from the ego vehicle to the goal point.
- change_left : Recalculates the route executing a left lane change.
- change_right : Recalculates the route executing a right lane change.

To execute a lane change, the LWP calculates a new route considering the ego vehicle's pose moved to the side lane as the first point of the route.

ROS publishers:

- route : Whenever a route is calculated, it is published in this topic as a t4ac_msg.msg/Path.
- lane_change_executed : This flag ease the control layer flow to assert when the lane change has been executed.

All the functionalities used in this part are developed in the map_utils.py file.

5.5. ROS Publishers and Subscribers

Although it has already been commented throughout the corresponding sections, here is a summary of the ROS publishers and subscribers of each module.

Map visualizator ROS publishers are shown in Table 5.1.

Map monitor ROS subscribers and publishers are shown in Table 5.2 and Table 5.3.

Monitor visualizator ROS subscribers and publishers are shown in Table 5.4 and Table 5.5.

Path planner ROS subscribers and publishers are shown in Table 5.6 and Table 5.7.

ROS Publishers	
Topic	**Msg type**
/t4ac/mapping/map/lanes_marker	visualization_msgs.msg.Marker

Table 5.1: Map visualizator ROS publishers

ROS Subscribers	
Topic	**Msg type**
/t4ac/planning/route	t4ac_msgs.msg.Path
/t4ac/localization/pose	nav_msgs.msg.Odometry

Table 5.2: Map monitor ROS subscribers

ROS Publishers	
Topic	**Msg type**
/t4ac/mapping/monitor/lanes	t4ac_msgs.msg.MonitorizedLanes
/t4ac/mapping/monitor/intersections	t4ac_msgs.msg.MonitorizedLanes
/t4ac/mapping/monitor/regElems	t4ac_msgs.msg.MonitorizedRegElem

Table 5.3: Map monitor ROS publishers

ROS Subscribers	
Topic	**Msg type**
/t4ac/mapping/monitor/lanes	t4ac_msgs.MonitorizedLanes
/t4ac/mapping/monitor/intersections	t4ac_msgs.MonitorizedLanes
/t4ac/mapping/monitor/regElems	t4ac_msgs.MonitorizedRegElem

Table 5.4: Monitor visualizator ROS subscribers

ROS Publishers	
Topic	**Msg type**
/t4ac/mapping/monitor/lanes_marker	visualization_msgs.msg.Marker
/t4ac/mapping/monitor/intersections_marker	visualization_msgs.msg.Marker
/t4ac/mapping/monitor/regElems_marker	visualization_msgs.msg.Marker

Table 5.5: Monitor visualizator ROS publishers

ROS Subscribers	
Topic	**Msg type**
/t4ac/localization/pose	Odometry
/t4ac/planning/goal	PoseStamped
/t4ac/decision_making/classic/t4ac_petrinets_ros/t4ac_petrinets_ros_node/change_left	Bool
/t4ac/decision_making/classic/t4ac_petrinets_ros/t4ac_petrinets_ros_node/change_right	Bool

Table 5.6: Path planner ROS subscribers

ROS Publishers	
Topic	**Msg type**
/t4ac/planning/route	Path
/t4ac/planning/lane_change_executed	Bool

Table 5.7: Path planner ROS publishers

Chapter 6

Results

In previous chapters we have seen all the process involved in the HD maps, from the map generation to the map monitor and path planning using the offline map (.xodr file).

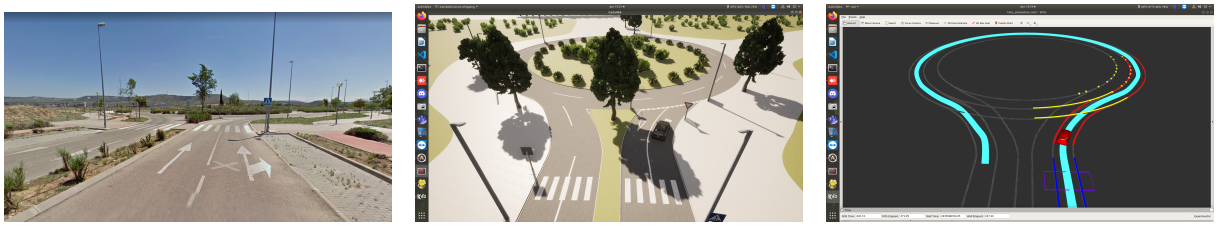
The first step has been generating the map files that can be imported in Carla Simulator and later used for mapping and planing tasks in both simulation and real cases (for these tasks only the .xodr is used).

Once we have the .xodr file (remember that this file is a text file describing the road map following the OpenDRIVE format), this file is parsed by the map parser, so the text file becomes useful information that can be used by other modules, as the mapping one and planning one.

The mapping module includes the map monitor functionality, that uses this offline map to generate a monitored area close to the ego-vehicle, easing the task of analyzing lanes, intersections and regulatory elements.

The planning module uses the offline map to generate a Directed Graph describing the road network. Then, it uses a global path planner based on A* algorithm that calculates waypoint routes from its current position to any other location of the map. The waypoints are always centered at the lanes, so the control module can use them to navigate through the road map.

Figure 6.1 shows how an area from our University Campus map has been generated and the used in the simulator, planning a route and monitoring the surrounding elements.



(a) Real (b) Simulator (c) RVIZ

Figure 6.1: Map generation and visualization from UAH campus image

The mapping and planning modules can work in both real and simulation cases in the same way, using always the same .xodr file.

In the following sections, three different examples of how the whole system works will be shown.

- Route in Carla Town03

- Route in UAH Campus
- Overtaking with lane change

In the first one, a route will be calculated in the Carla map Town03 . After that, a route will be calculated in our University Campus map generated with RoadRunner. Finally, a route executing a lane change for an overtaking will be shown. A video of the route execution in the simulator is included for each of the cases.

6.0.1. Route in Carla Town03

In Figure 6.2 we can observe a top view of the map in the simulator at the left, and the road map generated with the map parser and map monitor in RVIZ at the right.

In the RVIZ representation only driving lanes are represented, not considering other elements of the map environment.

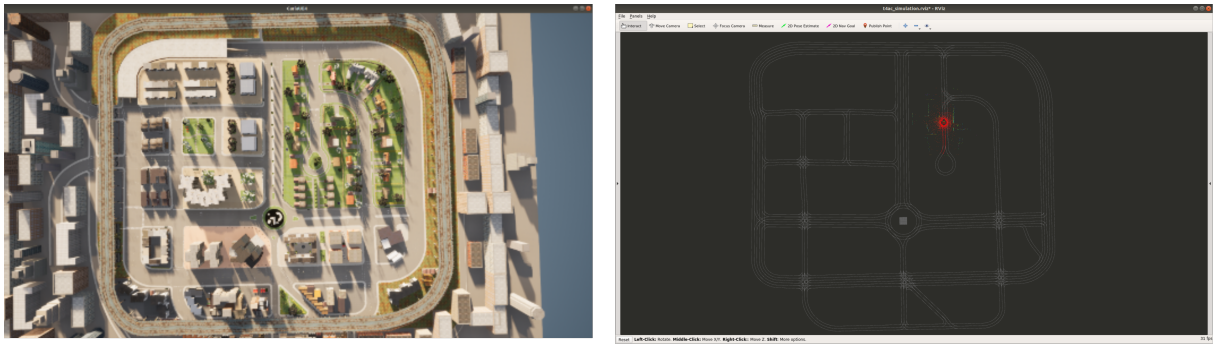


Figure 6.2: Top view of Town03 in CARLA Simulator (left) and RVIZ (right)

For this example, a route is launched using the planner developed in this project. We can observe the route in cyan from the starting to the goal position in Figure 6.3. Also relevant areas of intersection has been marked up in the image and will be hereafter shown.

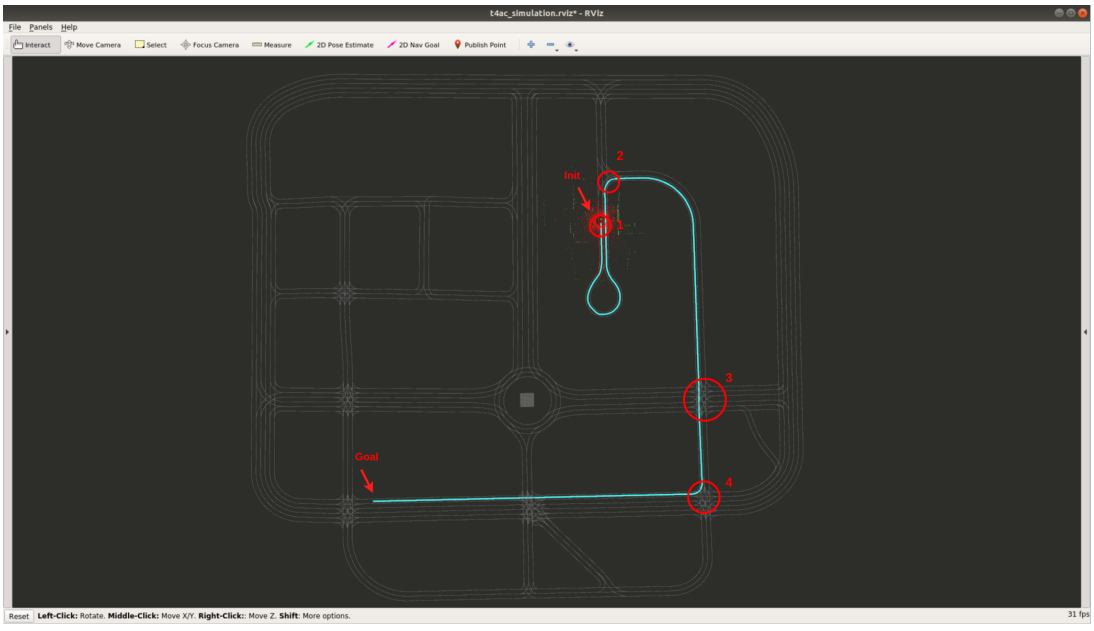
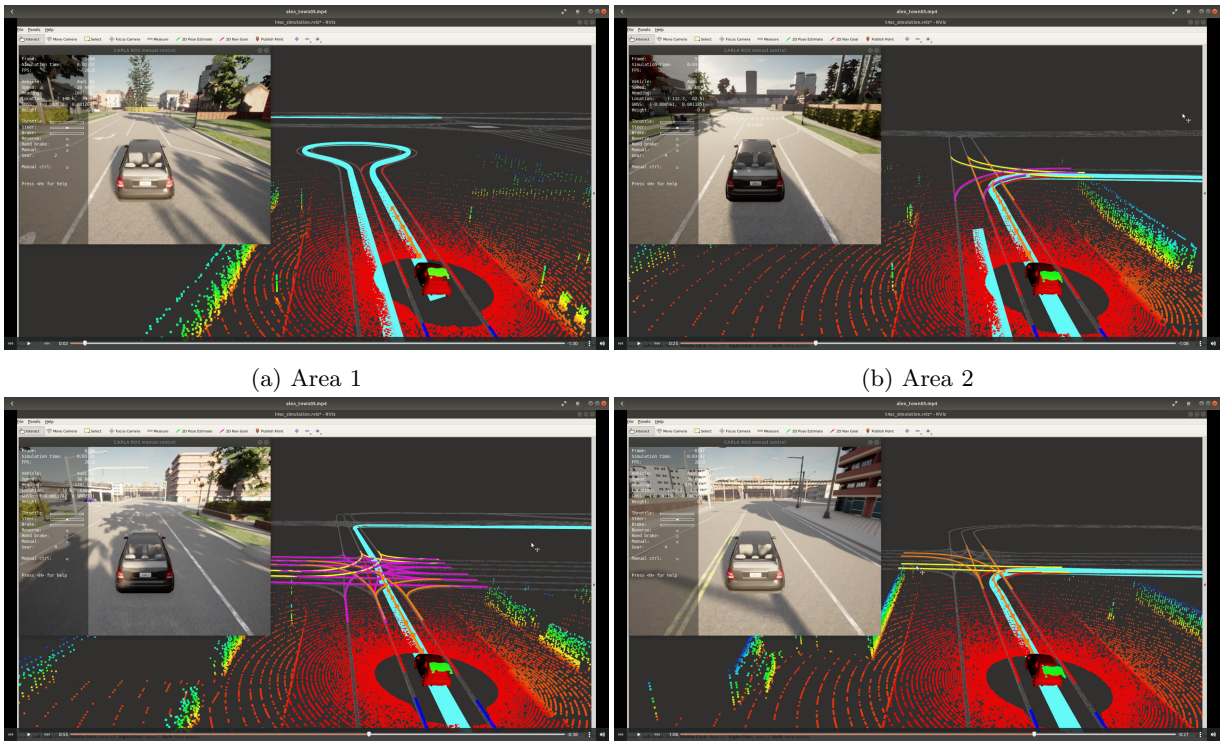


Figure 6.3: Top view of route launched in Town03

Figure 6.4 shows some snaps from the video in which can be seen how the map monitor represents the relevant lanes in intersections. The image is the RVIZ representation, and in the left upper corner is also a view of the simulator.

In the four images of Figure 6.4 can be observed how the route is represented in cyan, the current lane in red and the back lane in blue. In Areas 2, 3 and 4 can also be observed how the map monitor represents the intersection lanes: split lanes in orange, merging lanes in yellow and crossing lanes in purple. Even in Area 3 where multiple lanes appear in the same intersections, all of them are correctly monitored and represented. Reader can see the whole sequence in the following link: https://youtu.be/PUfUL4oLre8



(a) Area 1

(b) Area 2

(c) Area 3

(d) Area 4

Figure 6.4: Detail of areas in Figure 6.3

6.0.2. Route in Campus UAH

In this section we show another example of route performance, but in this case using the map we have created using RoadRunner of our University Campus.

This map is formed by the road-lanes map of the campus, but also some other elements have been added to simulate the real environment of the campus like buildings or trees.

Figure 6.5 shows a top view of the map in CARLA Simulator (left) and the road map representation in RVIZ (right).

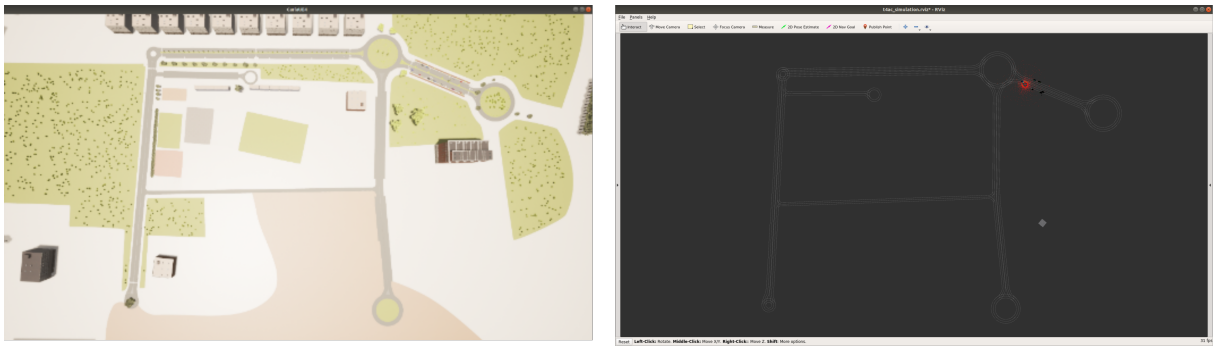


Figure 6.5: Top view of Campus UAH in CARLA Simulator (left) and RVIZ (right)

For this example, a route is launched using the planner developed in this project. We can observe the route in cyan from the starting to the goal position in Figure 6.6. Also relevant areas of intersection have been marked up in the image.

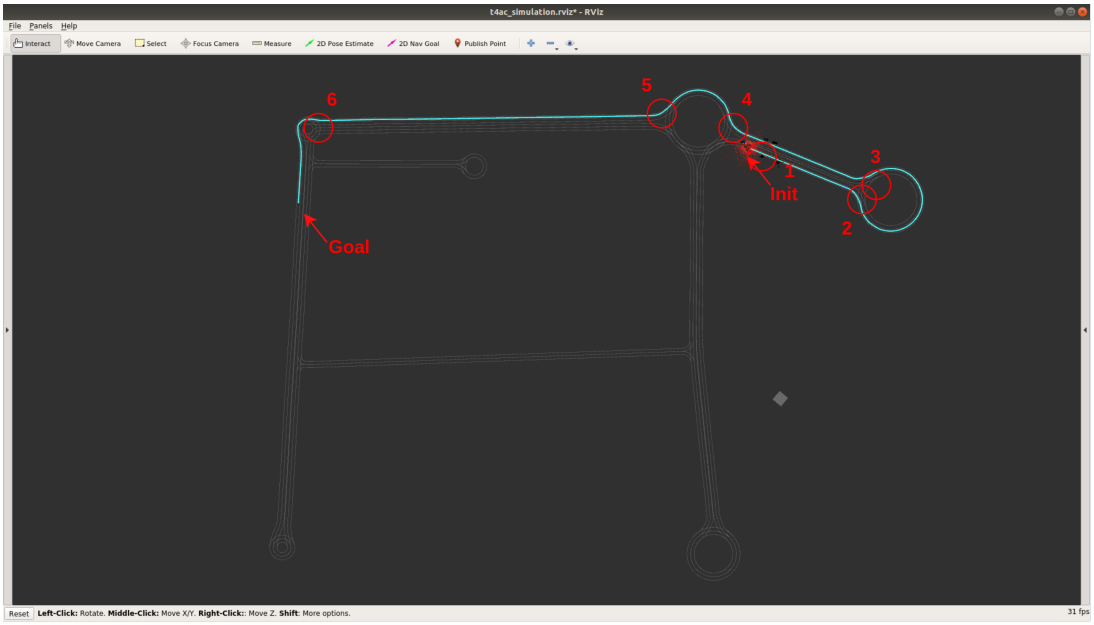


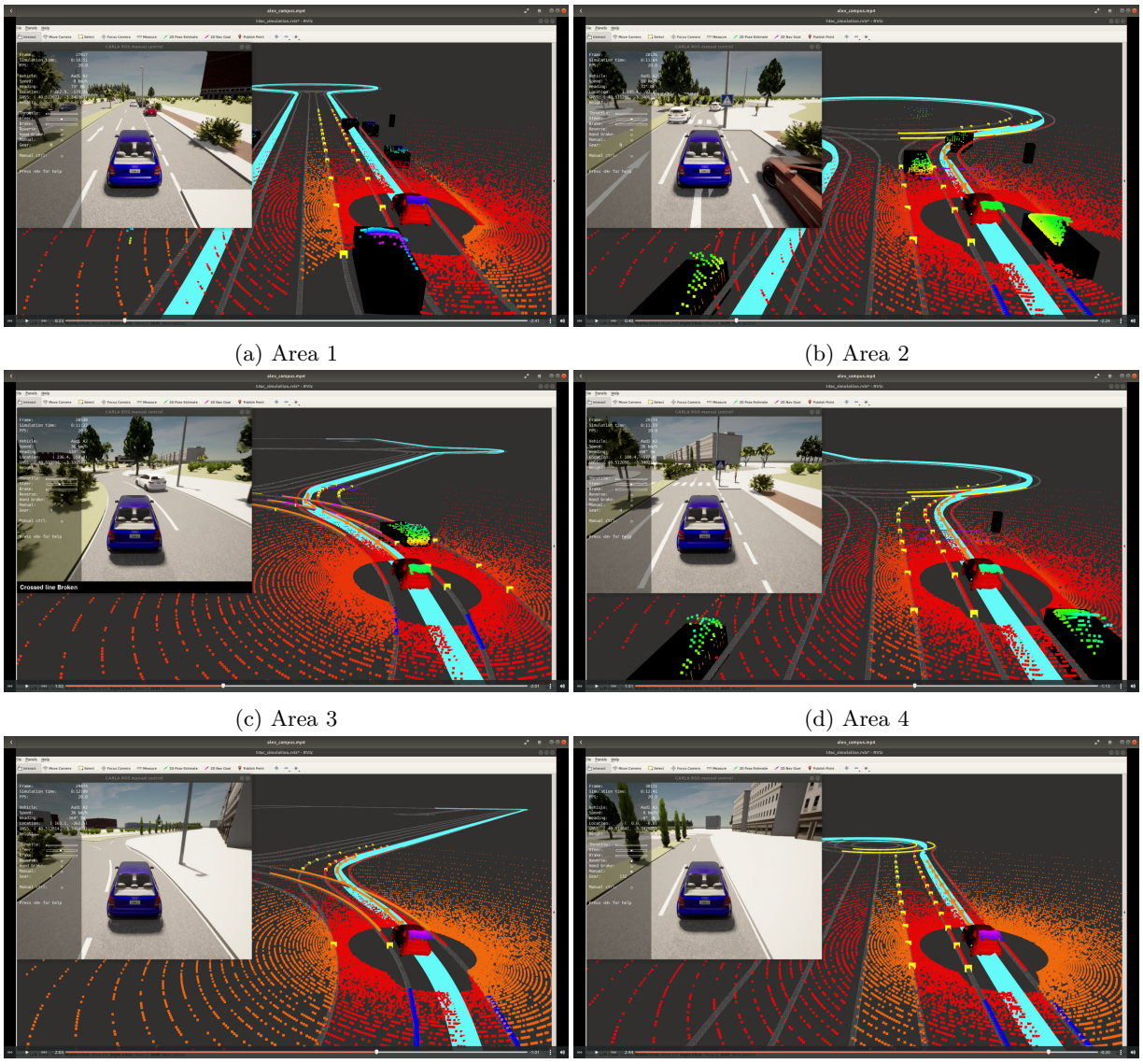
Figure 6.6: Top view of route launched in Campus UAH

In Figure 6.6 are some snaps from the video attached of the route in which can be shown how the map monitor represents the relevant lanes in intersections. The image is the RVIZ representation, and in the left upper corner is also a view of the simulator.

In the four images of Figure 6.7 can be observed how the route is represented in cyan, the current lane in red and the back lane in blue. The side lane is represented with yellow points when lane change is available because of the road marks. In some of the areas can also be observed how the map monitor represents the intersection lanes: split lanes in orange, merging lanes in yellow and crossing lanes in purple.

In areas 2 and 4 is also represented a crosswalk with a purple box. In the example of Town03 crosswalks were not considered because the XODR map file does not have crosswalk definitions.

The images of Figure 6.7 have also some black bounding boxes representing other vehicles, this is because during the route execution the perception layer was also activated. Reader can see the whole sequence in the following link: https://youtu.be/dSsV_nnDJcI



(a) Area 1 (b) Area 2

(c) Area 3 (d) Area 4

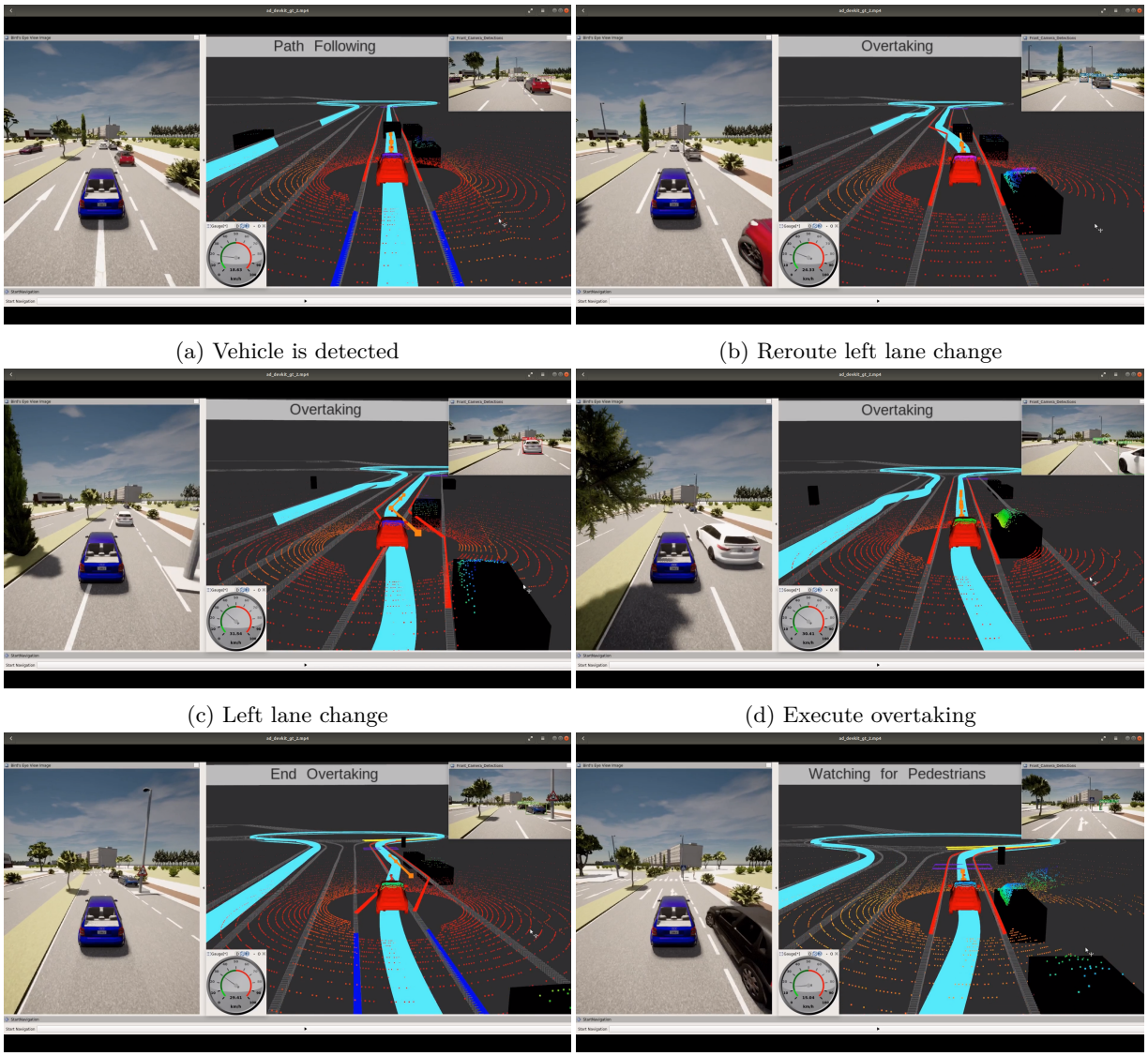
(e) Area 5 (f) Area 6

Figure 6.7: Detail of areas in Figure 6.6

6.0.3. Overtaking with lane change

Finally, a third video of an overtaking use case over the UAH Campus is attached. Figure 6.8 shows the different steps the path planning module follows, together with the decision making and the perception modules.

In first place the infront vehicle is detected, then the decision making module orders to execute an overtaking. When the overtaking is ordered, a new route is calculated changing to the left lane. When the ego-vehicle overpasses the infront vehicle, then a new route is calculated to change again to the right lane (ordered by the decision making module too). Reader can see the whole sequence in the following link: https://youtu.be/SbMEBw82RqA



(a) Vehicle is detected

(b) Reroute left lane change

(c) Left lane change

(d) Execute overtaking

(e) Reroute right lane change

(f) Return to the initial route

Figure 6.8: Overtaking with lane change

Chapter 7

Conclusions and future work

Finally, the conclusions after having finished this work are summarized. Also a section describing the possible future works related to this work is included.

7.1. Conclusions

The goal of this work was to develop a method to generate maps that can be used by the mapping and planning layers, both in real and simulation cases. Also, another goal was to obtain a system that works exactly in the same way in both simulation and real projects, avoiding the need of having two different methods depending on the case.

This way, the vehicle of the project can calculate paths that can be used by other layers for an autonomous navigation. The offline maps generated are used by the planning layer for path planning, but also by the mapping layer for generating useful data about the environment close to the vehicle that can be used by other layers as perception.

To achieve these goals, a prior study of the different existing map formats has been done, concluding that OpenDRIVE (.XODR) is the more appropriated format for our project. Because it allows to work the same way in both simulation and real cases, using exactly the same code.

For generating the XODR maps we have used the RoadRunner tool, that not only generates the XODR file for the mapping and planning layers but also generates the files needed for importing the map into the simulator.

Then, a map parser has been developed to get all the road map information from the XODR text file. This information is used by the map monitor to have offline information of the static map elements (lanes, intersections and regulatory elements) close to the ego vehicle.

The path planner also uses this information from the map parser, generating a directed graph that describes all the connections and costs of the road map at lane level. A global waypoint planner has been developed based on A* algorithm, able to generate routes from current position to any goal destination using the offline map. This planning also considers lane changing.

After having generated a relevant area of our University Campus map and having tested all this work in both simulation and real cases, we can conclude that goals proposed have been achieved.

7.2. Future work

This work has been completed and is being currently used in our project, but it can be continued and extended in multiple ways.

In the mapping layer, our current map generation process is valid, but it could be improved automating the process. A future work is to improve the method of map generation, obtaining an automatic HD map generation method using the ego-vehicle and its sensors to map the environment. It could be useful not only for generating new maps, but then for updating the existing maps with real time road changes like streets temporarily closed or accident on the road.

The map monitor can be extended, adding more elements to be considered like regulatory elements that are not currently included in the monitor.

The visualization part can also be improved, getting a more realistic environment representation in RVIZ visualizer.

About the path planning, the ideal would be to have two planners: global and local planner. Currently only a global planner is doing all the path planning task, but it is totally dependent of the precision with which the map has been defined and the precision of the localization module. A future improvement would be to have a global planner based on the HD map that only generates a topological route at road-lane level, and a local planner based on perception that identifies the lanes in real time and generates the waypoints needed by the control module to navigate. This way, we would reach a much more robust and complete system.

Bibliography

[1] S. Casas, A. Sadat, and R. Urtasun, "Mp3: A unified model to map, perceive, predict and plan," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[2] J. hwan Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *2013 American control conference*. IEEE, 2013, pp. 188–193.

[3] S. Song, "Towards autonomous driving at the limit of friction," Master's thesis, University of Waterloo, 2015.

[4] H. G. Seif and X. Hu, "Autonomous driving in the icity-hd maps as a key challenge of the automotive industry," *Engineering*, vol. 2, no. 2, pp. 159–162, 2016.

[5] F.-A. Moreno, J. Gonzalez-Jimenez, J.-L. Blanco, and A. Esteban, "An instrumented vehicle for efficient and accurate 3d mapping of roads," *Computer-Aided Civil and Infrastructure Engineering*, vol. 28, no. 6, pp. 403–419, 2013.

[6] M. Elhousni, Y. Lyu, Z. Zhang, and X. Huang, "Automatic building and labeling of hd maps with deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 08, 2020, pp. 13 255–13 260.

[7] "Roadrunner," https://es.mathworks.com/products/roadrunner.html [Last checked 21/june/2021].

[8] J. Godoy, A. Artuñedo, and J. Villagra, "Self-generated osm-based driving corridors," *IEEE Access*, vol. 7, pp. 20 113–20 125, 2019.

[9] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org, 2017.

[10] "Asam opendrive," https://www.asam.net/standards/detail/opendrive/ [Last checked 21/june/2021].

[11] "Josm (java openstreetmap editor)," https://josm.openstreetmap.de/ [Last checked 21/june/2021].

[12] "Asam e.v. adopts standards for highly automated driving," https://www.asam.net/news-media/press-releases/detail/?tx_news_pi1%5Bnews%5D=48&cHash=726c911525f72ede5ade9e755c3c8369 [Last checked 21/june/2021].

[13] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[14] "Nvidia drive mapping," https://developer.nvidia.com/drive/drive-mapping [Last checked 21/june/2021].

[15] "Here hd live map," https://www.here.com/platform/automotive-services/hd-maps [Last checked 21/june/2021].

[16] "liblanelet," https://github.com/fzi-forschungszentrum-informatik/liblanelet [Last checked 22/june/2021].

[17] "Pythonapi," https://carla.readthedocs.io/en/latest/python_api/ [Last checked 22/june/2021].

[18] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 420–425.

[19] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 1672–1679.

[20] "Lanelet1 vs lanelet2," https://github.com/fzi-forschungszentrum-informatik/Lanelet2/blob/master/lanelet2_core/doc/Lanelet1Compability.md [Last checked 07/sept/2021].

[21] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[22] H.-P. Huang and S.-Y. Chung, "Dynamic visibility graph for path planning," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2813–2818.

[23] Y. Hwang and N. Ahuja, "A potential field approach to path planning," *IEEE Trans. Robotics Autom.*, vol. 8, pp. 23–32, 1992.

[24] P. Bhattacharya and M. L. Gavrilova, "Voronoi diagram in optimal path planning," in *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*. IEEE, 2007, pp. 38–47.

[25] J. Borenstein, Y. Koren *et al.*, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE transactions on robotics and automation*, vol. 7, no. 3, pp. 278–288, 1991.

[26] I. Ulrich and J. Borenstein, "Vfh+: Reliable obstacle avoidance for fast mobile robots," in *Proceedings. 1998 IEEE international conference on robotics and automation (Cat. No. 98CH36146)*, vol. 2. IEEE, 1998, pp. 1572–1577.

[27] ——, "Vfh/sup*: Local obstacle avoidance with look-ahead verification," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 3. IEEE, 2000, pp. 2505–2511.

[28] R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *Proceedings of IEEE international conference on robotics and automation*, vol. 4. IEEE, 1996, pp. 3375–3382.

[29] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[30] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proceedings 1999 ieee international conference on robotics and automation (Cat. No. 99CH36288C)*, vol. 1. IEEE, 1999, pp. 341–346.

[31] J. Minguez and L. Montano, "Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 45–59, 2004.

[32] J. W. Durham and F. Bullo, "Smooth nearness-diagram navigation," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 690–695.

[33] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms," in *Algorithmics of large and complex networks*. Springer, 2009, pp. 117–139.

[34] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route planning in transportation networks," in *Algorithm engineering*. Springer, 2016, pp. 19–80.

[35] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[36] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory." in *SODA*, vol. 5. Citeseer, 2005, pp. 156–165.

[37] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[38] D. Van Vliet, "Improved shortest path algorithms for transport networks," *Transportation Research*, vol. 12, no. 1, pp. 7–20, 1978.

[39] L. Fu, D. Sun, and L. R. Rilett, "Heuristic shortest path algorithms for transportation applications: State of the art," *Computers & Operations Research*, vol. 33, no. 11, pp. 3324–3343, 2006.

[40] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter, "Exact routing in large road networks using contraction hierarchies," *Transportation Science*, vol. 46, no. 3, pp. 388–404, 2012.

[41] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *SIAM Journal on Computing*, vol. 32, no. 5, pp. 1338–1355, 2003.

[42] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2015.

[43] A. Stentz, "Optimal and efficient path planning for partially known environments," in *Intelligent unmanned ground vehicles*. Springer, 1997, pp. 203–220.

[44] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The field d* algorithm," *Journal of Field Robotics*, vol. 23, no. 2, pp. 79–101, 2006.

[45] A. Nash, K. Daniel, S. Koenig, and A. Felner, "Theta^*: Any-angle path planning on grids," in *AAAI*, vol. 7, 2007, pp. 1177–1183.

[46] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643, 2008.

[47] M. Pivtoraiko and A. Kelly, "Efficient constrained path planning via search in state lattices," in *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*. Munich Germany, 2005, pp. 1–7.

[48] Q. Li, Z. Zeng, B. Yang, and T. Zhang, "Hierarchical route planning based on taxi gps-trajectories," in *2009 17th International Conference on Geoinformatics*. IEEE, 2009, pp. 1–5.

[49] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke *et al.*, "Junior: The stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[50] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 1879–1884.

[51] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014.

[52] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[53] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[54] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[55] J. Horst and A. Barbera, "Trajectory generation for an on-road autonomous vehicle," in *Unmanned systems technology VIII*, vol. 6230. International Society for Optics and Photonics, 2006, p. 62302J.

[56] M. Brezak and I. Petrović, "Real-time approximation of clothoids with bounded error for path planning applications," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 507–515, 2013.

[57] D. J. Walton and D. S. Meek, "A controlled clothoid spline," *Computers & Graphics*, vol. 29, no. 3, pp. 353–363, 2005.

[58] S. Glaser, B. Vanholme, S. Mammar, D. Gruyer, and L. Nouveliere, "Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction," *IEEE Transactions on intelligent transportation systems*, vol. 11, no. 3, pp. 589–606, 2010.

[59] P. Petrov and F. Nashashibi, "Modeling and nonlinear adaptive control for autonomous vehicle overtaking," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 4, pp. 1643–1656, 2014.

[60] D. Walton, D. Meek, and J. Ali, "Planar g2 transition curves composed of cubic bézier spiral segments," *Journal of Computational and Applied Mathematics*, vol. 157, no. 2, pp. 453–476, 2003.

[61] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, D. Anderson *et al.*, "Odin: Team victortango's entry in the darpa urban challenge," *Journal of field Robotics*, vol. 25, no. 8, pp. 467–492, 2008.

[62] A. Piazzi, C. L. Bianco, M. Bertozzi, A. Fascioli, and A. Broggi, "Quintic g/sup 2/-splines for the iterative steering of vision-based autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 3, no. 1, pp. 27–36, 2002.

[63] Z. Shiller, Y.-R. Gwo *et al.*, "Dynamic motion planning of autonomous vehicles," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, pp. 241–249, 1991.

[64] T. Berglund, A. Brodnik, H. Jonsson, M. Staffanson, and I. Soderkvist, "Planning smooth and obstacle-avoiding b-spline paths for autonomous mining vehicles," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 1, pp. 167–172, 2009.

[65] L. Romani and M. A. Sabin, "The conversion matrix between uniform b-spline and bézier representations," *Computer aided geometric design*, vol. 21, no. 6, pp. 549–560, 2004.

[66] J. Funke, P. Theodosis, R. Hindiyeh, G. Stanek, K. Kritatakirana, C. Gerdes, D. Langer, M. Hernandez, B. Müller-Bessler, and B. Huhnke, "Up to the limits: Autonomous audi tts," in *2012 IEEE Intelligent Vehicles Symposium*. IEEE, 2012, pp. 541–547.

[67] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2061–2067.

[68] D. González, J. Pérez, R. Lattarulo, V. Milanés, and F. Nashashibi, "Continuous curvature planning with obstacle avoidance capabilities in urban scenarios," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2014, pp. 1430–1435.

[69] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller *et al.*, "Making bertha-drive an autonomous journey on a historic route," *IEEE Intelligent transportation systems magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[70] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The international journal of robotics research*, vol. 29, no. 5, pp. 485–501, 2010.

[71] L. Caltagirone, M. Bellone, L. Svensson, and M. Wahde, "Lidar-based driving path generation using fully convolutional neural networks," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017, pp. 1–6.

[72] C. Gomez-Huelamo, J. Del Egido, L. M. Bergasa, R. Barea, M. Ocana, F. Arango, and R. Gutierrez-Moreno, "Real-time birdâs eye view multi-object tracking system based on fast encoders for object detection," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–6.

[73] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[74] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[75] C. Gomez-Huelamo, J. Del Egido, L. M. Bergasa, R. Barea, E. Lopez-Guillen, F. Arango, J. Araluce, and J. Lopez, "Train here, drive there: Simulating real-world use cases with fully-autonomous driving architecture in carla simulator," in *Workshop of Physical Agents*. Springer, 2020, pp. 44–59.

[76] R. Gutierrez, E. Lopez-Guillen, L. M. Bergasa, R. Barea, O. Perez, C. Gomez-Huelamo, F. Arango, J. Del Egido, and J. Lopez-Fernandez, "A waypoint tracking controller for autonomous road vehicles using ros framework," *Sensors*, vol. 20, no. 14, p. 4062, 2020.

[77] M. Tradacete, A. Saez, J. F. Arango, C. G. Huelamo, P. Revenga, R. Barea, E. Lopez-Guillen, and L. M. Bergasa, "Positioning system for an electric autonomous vehicle based on the fusion of multi-gnss rtk and odometry by using an extented kalman filter," in *Workshop of Physical Agents*. Springer, 2018, pp. 16–30.

[78] J. F. Arango, L. M. Bergasa, P. A. Revenga, R. Barea, E. López-Guillén, C. Gómez-Huélamo, J. Araluce, and R. Gutiérrez, "Drive-by-wire development process based on ros for an autonomous electric vehicle," *Sensors*, vol. 20, no. 21, p. 6121, 2020.

[79] P. Green, "Where do drivers look while driving (and for how long)," *Human factors in traffic safety*, vol. 2, pp. 77–110, 2002.

[80] "Distancias seguras de frenado," https://cdn3.capacitateparaelempleo.org/assets/76k9ldq.pdf [Last checked 23/june/2021].

[81] "Ros wiki," http://wiki.ros.org/ [Last checked 03/sept/2021].

[82] "Docker," https://www.docker.com/ [Last checked 03/sept/2021].

[83] "Asam opendrive 1.6," https://releases.asam.net/OpenDRIVE/1.6.0/ASAM_OpenDRIVE_BS_V1-6-0.html [Last checked 7/july/2021].

[84] "Unreal engine," https://www.unrealengine.com/en-US/ [Last checked 12/july/2021].

[85] "Networkx - network analysis in python," https://networkx.org/ [Last checked 30/july/2021].

Appendix A

Budget

This appendix details the main hardware, software and human resources used in this project.

A.1. Hardware

Name	Cost [€]
Desktop laboratory PC	2990
DGNNS	6000
Total	**8990**

Table A.1: Hardware

A.2. Software

Name	Cost [€]
RoadRunner	-
Ubuntu 18.04	-
VSCode	-
ROS Melodic	-
CARLA Simulator	-
Total	**0**

Table A.2: Software

A.3. Professional fees

Professional fees	Months	Cost/month [€]	Total cost [€]
Engineering	7	1400	9800
Typing	1	1400	1400
Total			**11200**

Table A.3: Professional fees

A.4. Execution budget

Name	Cost [€]
Hardware	8990
Software	0
Personal fees	11200
Total	**20190**

Table A.4: Execution budget

A.5. Total costs

The final budget for this Master's Thesis ends with the application of VAT, stipulated in 21 %, based on the execution budget per contract.

Name	Cost [€]
Execution budget	20190
VAT (21 %)	4239.9
Total	**24429.9**

Table A.5: Total costs

The final estimated budget for the completion of this Master's Thesis amounts to a total of 24429.9 €(twenty four thousand four hundred and twenty nine euros and ninety cents).

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá